elektor books

Control Your Home with Raspberry Pi

Secure, Modular, Open-Source and Self-Sufficient

Koen Vervloesem

elektor
design > share > sell

Control Your Home with Raspberry Pi

Koen Vervloesem

an Elektor Publication

LEARN DESIGN SHARE

● This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
78 York Street
London W1H 1DP, UK
Phone: (+44) (0)20 7692 8344

First published in the United Kingdom 2020

● British Library Cataloguing in Publication Data
Catalogue record for this book is available from the British Library

● **ISBN 978-1-907920-94-3**

● **EISBN 978-3-89576-383-0**

● **EPUB 978-3-89576-382-3**

Prepress production: DMC ¦ daverid.com
Printed in the Netherlands by Wilco

Images and logos used in this book are courtesy of Material Design Icons, Cisco, and the Raspberry Pi Foundation

Table of Contents

• Preface

Ever since the Raspberry Pi was introduced, the popular single-board computer has been used by enthusiasts to automate their home. That's not a coincidence: the Raspberry Pi is a powerful computer in a small package, with lots of interfacing options to control various devices.

In this book, I'll show you how you can automate your home with a Raspberry Pi. You can do this in many ways and with various software and hardware choices. I'll show you one way, which is a bit different from what you'll read in many other books, but my approach has its merits, and I'll explain why.

You'll learn how to use various wireless protocols for home automation, such as Bluetooth, 433.92 MHz radio waves, Z-Wave, and Zigbee. Soon you'll automate your home with Python, Node-RED, and Home Assistant, and you'll even be able to speak to your home automation system. All of this in a secure way, with a modular system, completely open-source and without relying on third-party services.

At the end of the book, you can install and configure your Raspberry Pi as a highly flexible home automation gateway for your protocols of choice and link various services with MQTT to make it a system of your own. This DIY (do it yourself) approach is a bit more laborious than just installing an off-the-shelf home automation system, but in the process, you learn a lot, and in the end, know exactly what's running your house and how to tweak it. And that's why you were interested in the Raspberry Pi in the first place, right?

Koen Vervloesem, May 2020

Chapter 1 • Introduction

In this introductory chapter, I give a short overview of what home automation is and why you would use a Raspberry Pi as a home automation gateway. Then I describe what I consider properties of a 'good' home automation system:

- secure
- modular
- open-source
- self-sufficient

In the rest of this book, I'll explain step by step how to create such a good home automation system with a Raspberry Pi.

1.1 • What is home automation?

Home automation is the process or result of automating systems that are running at home: lighting, HVAC (heating, ventilation, and air conditioning), appliances such as washing machines, blinds, and roller shutters, and so on. A home automation system is also able to use information from environmental sensors (temperature, humidity, pressure, CO2, …), smart meters, movement sensors, presence sensors, cameras, and so on.

A home automation system typically consists of:

- a central gateway (also called controller or hub), which controls devices and reads sensor measurements
- controllable devices
- sensors

The controllable devices and sensors are regularly called "smart devices", although almost none of them are really smart. Another name you'll see for them is IoT (Internet of Things) devices because they can be (directly or indirectly) linked to and controlled over the internet.

Figure 1.1 A home automation system consists of a central gateway and various controllable devices and sensors.

A home automation gateway always has a user interface. Of course, the purpose of home automation is to automate as much as possible, so the idea is that the user shouldn't have to use this user interface that much. But a user interface is still essential to:

- configure the home automation gateway: for instance if the sun goes down, close the blinds;
- manually control your devices: this should still be possible because you can't automate everything;
- show you a nice dashboard of sensor measurements: for instance to see the inside and outside temperature.

This user interface can come in many forms:

- Most home automation gateways have a web server running, which supplies a web interface as the user interface. You can access this web interface on any computer or mobile device.
- Some systems have a mobile app for Android or iOS, which is generally better adapted to the specific requirements of mobile devices, such as a smaller screen.
- It's also possible to use a dedicated touch screen, for instance hanging on the wall, to control your home automation system, and to show you some nice graphs.[1]
- Last but not least, in recent years home automation systems have added a new kind of user interface: speech. With a so-called voice assistant or smart assistant (again, they are not that smart), you can give speech commands to your home automation system and it replies with spoken messages.

This book is not focused on any of these user interfaces; it's more about the backend services and how to link and automate them. However, I cover two home automation projects with a web interface in Chapter 10 (Home Assistant and Node-RED), and create a voice assistant for your home automation system with Rhasspy (Chapter 12). You should consult the documentation of these projects if you're more interested in the user interface side of home automation.

1.2 ● Why use a Raspberry Pi as a home automation gateway?

If you buy an off-the-shelf home automation system, the gateway is a small box that looks somewhat like a router or a Wi-Fi access point. In this book, we'll show you how you can create your own home automation gateway with a Raspberry Pi.

But why would you do that? Because you can, of course! More seriously, the Raspberry Pi is what makes the approach in this book possible. We'll go into the advantages of this approach in the next section, but the number one reason to use a Raspberry Pi as your home automation gateway is: you are in control.

An off-the-shelf home automation gateway isn't flexible: you can only do with it what the

1 In many cases this touch screen is just running a fullscreen web browser visiting the home automation gateway's embedded web server.

manufacturer allows, you rely on the manufacturer's goodwill to receive updates and new functionality, and most of the time you can't "hack" on it yourself.[2]

Contrast this with the Raspberry Pi. You can choose your operating system (which we'll do in the next chapter), you can choose which communication protocols you'll want to support (which we'll cover in various chapters in this book), you can choose your user interface, and so on. You can even choose the case to protect your Raspberry Pi and which expansion boards you connect, as there's a whole ecosystem of hardware for the Raspberry Pi.

Of course, you can also run home automation software on a more powerful system, such as a NAS (network-attached storage) or a home server. But the Raspberry Pi has several advantages to these systems:

- It's very low-power, so it doesn't cost you much to keep it running 24/7.
- For most home automation tasks you don't need the processing power that these other systems offer.
- The ecosystem of software and hardware for the Raspberry Pi is immense, as well as the number of resources where you find more information about it.[3] This is also a reason to choose the Raspberry Pi over similar single-board computers from other manufacturers.

1.3 ● The properties of a good home automation system

A good home automation system should:

- be secure, so you don't risk someone else controlling your house or spying on you at home;
- be modular, to make it easy to plug in other protocols or applications;
- only use open-source software;
- be self-sufficient, not relying on cloud systems from Google, Amazon, or other parties.

This is my highly opinionated vision, and it's this vision that I build upon in this book.
If you consider these properties for a moment, you'll see that this vision is almost diametrically opposed to most off-the-shelf systems you'll find. I'll give some examples in the next subsections.

It's possible that you don't agree with some of these properties, or that you don't have such strong feelings about them as I do. That's OK: while I explain an approach in this book

2 With hacking I don't mean gaining unauthorized access to a computer (which is the connotation the word unfortunately has received). The Jargon File describes a hacker as "a person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary." (The Jargon File, http://www.catb.org/jargon/html/H/hacker.html)

3 For instance, the Raspberry Pi Foundation publishes its official magazine about the Raspberry Pi, MagPi (https://magpi.raspberrypi.org), and Elektor (the publisher of this book) publishes Dutch (https://www.magpi.nl) and French (https://www.magpi.fr) editions of the magazine.

that implements this vision, thanks to its modularity you can certainly plug in proprietary software or cloud systems if you prefer these. Heck, you can even add monolithic and insecure software, but I don't tell you how.

In the next subsections, I'll go over these four properties in more detail, and I hope that at the end of this chapter you'll agree with me that this approach to home automation is the right one.

1.3.1 ● Secure

Of course, a home automation system should be secure. No one can be against it, can't they? A home automation system controls your home, so whoever can break into it can make your life very miserable.

Unfortunately, even if a manufacturer tells you that his system is secure, chances are that it isn't. Security is very difficult to attain, and most manufacturers don't want to spend the resources needed to secure their system.[4]

Home automation and IoT devices are notoriously insecure. At the Usenix Security Conference 2019, the Czech security software company Avast and Stanford University presented their research of household IoT devices. Avast scanned 83 million IoT devices in 16 million homes around the world of people who agreed to share these data. The results of the study published in "All Things Considered: An Analysis of IoT Devices on Home Networks" (https://press.avast.com/hubfs/stanford_avast_state_of_iot.pdf) were staggering:

- 7% percent of all IoT devices support an obsolete, insecure, and completely unencrypted protocol such as Telnet or FTP.
- Of these, 17% exhibit weak FTP passwords, and 2% have weak Telnet passwords.
- Surveillance cameras have the weakest Telnet profile, with more than 10% of them that support Telnet with weak credentials.
- 3% percent of the homes are externally visible on the internet and more than half of those have a known vulnerability or a weak password.

This is not an isolated study. Not a week goes by without some news items about insecure devices, most of the time because basic security measures such as strong passwords are not enforced by the manufacturer or basic programming errors have been made. To give you an idea about what can happen: in 2018 nude videos of the Dutch women's handball team appeared on a popular porn website because the surveillance cameras of the dressing room of a sauna were broken into. Imagine if someone can access your baby monitor with a camera or your security camera in your living room or bedroom…

So what can you do to secure your home automation system? If you choose an off-the-shelf system: not much. You fully rely on the manufacturer's ability to create a secure

4 Most consumers probably wouldn't want to pay more for a secure home automation system anyway.

system and the goodwill to keep supplying patches that solve security issues that have been discovered. And the home automation and IoT industries have clearly shown they are not up to the task. This is one of the reasons why I prefer open-source software. Not because it is always secure, but because the transparency of the open-source development process forces developers to create more secure software.

Security is such an important property of a home automation system that I dedicate an entire chapter in this book about it. It's such a vast topic that entire books are written about it, and I encourage you to read much more about computer security than I can tell you here. In Chapter 3 I'll cover the most important tools you need to secure your home automation system, so you don't need to be paranoid and continuously think about the possibility that someone is currently spying on you.

1.3.2 ● Modular

There are many competing standards and communication protocols for home automation, such as Z-Wave, Zigbee, and KNX. Other protocols aren't specific to home automation but are very usable in this domain too, such as Wi-Fi, Bluetooth, or Near Field Communication (NFC).

Unfortunately, many off-the-shelf home automation gateways support only a small subset of these protocols or even use a proprietary protocol that locks you into using devices of the same manufacturer. That severely limits your choice of products.

You can't know which protocols will become popular in a few years, and maybe you like one product that uses Z-Wave and another product that uses Zigbee. It should be easy to interconnect these devices, even when they use different protocols.

This is why a good home automation system should be modular, which makes it possible to plug in new components when you want to support a new protocol, add a new user interface or extend its functionality in another way.

Many of the wireless communication protocols for home automation need a dedicated transceiver because they work on a specific radio frequency. That's where the Raspberry Pi shines: you can easily connect Z-Wave, Zigbee, or 433.92 MHz transceivers using the USB ports or the GPIO header. So you can start with a basic Raspberry Pi setup supporting only IoT devices that are communicating over Wi-Fi and Bluetooth, add an RTL-SDR USB dongle to read the measurements of your 433.92 MHz weather sensors, later add a Z-Wave HAT on the board when you start adding Z-Wave sensors to your house and then add a Zigbee USB transceiver when you want to control some Zigbee lights.

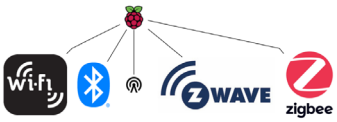

Figure 1.2 A good home automation system is modular enough to support many home automation protocols.

Modularity is also important for software. There's a lot of user-friendly software to make your Raspberry Pi a home automation gateway.[5] So you just install this software on your Raspberry Pi and that's it: you have a gateway that supports several devices. Some of these systems are very modular and extensible, others aren't. Many of them support MQTT (Message Queuing Telemetry Transport), a common language to exchange messages.

MQTT has become the standard for interoperability between various home automation devices. For instance, if your home automation gateway of choice doesn't support Zigbee but it does support MQTT, then you only have to run the Zigbee2mqtt software (see Chapter 9), which translates the Zigbee protocol to MQTT messages. Your gateway can then talk to your Zigbee devices using MQTT.

Modularity also means that you don't have to have one gateway. You can perfectly have your main gateway in your basement, but install a second gateway with your 433.92 MHz receiver for your environmental sensors in your living room because that gives you better coverage to receive data from these wireless sensors. If you're using MQTT, that's very simple to implement: you just relay the sensor readings that your gateway in the living room receives to your MQTT broker, after which your main gateway receives the readings in the MQTT format.

In short: a good modular home automation system means that you can mix and match the devices that you like, irrespective of their protocol, and you can use the software and hardware components of your choice, in various locations in your house.

1.3.3 ● Open-Source

Source code is code written in a human-readable programming language, that specifies the actions a computer has to perform. The source code of a program is then compiled to machine code that the computer can execute, or it's interpreted on the fly to machine code and thus immediately executed by the computer.

Most software is being distributed as machine code, so it's not readable for us. If you buy an off-the-shelf home automation gateway, you generally don't get access to its source code, so you cannot peek into it to see what it does or to assess its quality. You just have to believe the manufacturer on his word. Is that enough for you if it's about software that will get to know you intimately because it processes sensor readings and even camera images about you in your home? Not for me.

But there's a type of software where you do get access to the source code: free and open-source software (sometimes abbreviated as FOSS or even FLOSS).[6] If you really want to be precise, there's free software and open-source software, but for the end-user, the differences are minimal and mostly philosophical. When I talk about "open-source

5 An example is Home Assistant, which I will introduce in Chapter 10.

6 The L in FLOSS stands for "libre", which is kind of a synonym to "free" but making it clearer that it's about maintaining the user's civil liberties: "free" as in "free speech", not as in "free beer", as they say.

software" in this book, I mean free and open-source software.

But what when you're not a programmer and don't even understand the source code of your home automation system? Even then the use of open-source software has a lot of advantages. It's not because you don't have the programming experience that others can't help. Most open-source projects have a decentralized software-development model that encourages open collaboration.

So if you find a bug in the software, or see something wrong in its source code but don't have the programming experience needed to fix it, just report the bug on the issue tracker of the project, and hopefully, someone else in the project's community will step in and fix it. It all depends on the health of the project's community. But a good open-source project has a vibrant community of developers and users who collaborate to continuously improve the software.

In the process of writing this book, I participated in various communities of the programs I covered. I opened issues to report bugs, fixed some bugs, added support for new devices, contributed documentation, and helped people build their software for Raspberry Pi. This was all only possible because they are open-source.

And open-source doesn't just mean getting access to the source code, it's much more than that. If you want to get an idea about what open-source is, I recommend you to read the Open Source Definition on https://opensource.org by the Open Source Initiative. For instance, with open-source software you are not only able to read its source code, you are also allowed to modify it and distribute your modifications.

Even more, when you know a specific software project is open-source, you know that it doesn't arbitrarily restrict what you can do with it. The Open Source Definition even explicitly lists that the license of open-source software must not discriminate against any person or group of persons, nor restrict anyone from making use of the program in a specific field of endeavor.

Open-source is a complex and nuanced topic, and I recommend you to read about it more if you're not accustomed to it. The decentralization and transparency of the open-source development model gives power back to the users, where it belongs. For home automation that's even more important. I don't want a company having control over my house. That's why you'll see that all software in this book is open-source.

1.3.4 ● Self-sufficient

During the last ten years, there has been a worrying development in the computer industry: we all depend more and more on (centralized) cloud systems. Unfortunately, the home automation industry didn't escape this fate. Many popular home automation and IoT systems depend on a cloud server. Some examples:

- the Ring video doorbell with Wi-Fi camera;

- the Nest Learning Thermostat;
- so-called 'smart speakers' running voice assistants, like Amazon Echo and Google Home;
- the IFTTT service that links various other services.

This isn't without its problems. In the last couple of years a couple of cloud services for home automation stopped working for their users:

Revolv Hub
In 2014 Nest bought the company that was selling the Revolv Hub home automation system, not long after Nest itself was acquired by Google. In 2016 Nest shut down the servers Revolv Hub depended on, after announcing it with a quiet note on the website of Revolv a few months earlier. That meant that the $300 Revolv Hub ceased functioning entirely.

Best Buy Insignia devices
At the end of 2019, Best Buy announced that several of their Insignia-branded smart devices would stop working because they decided to shut down the corresponding backend systems.

Wink devices
In May 2020 Wink (with the catchphrase "A simpler way to a smarter home") announced with just a week's notice that it would start charging a monthly fee for the use of their services. Users that didn't want to pay were no longer be able to access their Wink devices and their automations were disabled. The Wink Hub, that had been in stores with the clear description "no required monthly fees, ever", was rendered useless. Ironically, the announcement ended with the message "Our user community is integral to Wink, and we want to continue to be your trusted smart home provider."

Now imagine when the highly popular IFTTT would stop their service: suddenly the automations of millions of people would stop working.

It also just makes no sense to use cloud services to automate your home. Home automation comes down to: you want one device to be able to respond to another device in your home. For instance: the motion sensor in your bathroom detects motion at night, and this turns the bathroom light on for five minutes. If you use IFTTT to link both devices, the motion sensor has to send a message to the internet, IFTTT relays the message to your bathroom light (for instance a Philips Hue light), and the bathroom light turns on.

But there's no need for an internet service like IFTTT between both devices because they both are in your home, so it makes much more sense to link them locally, using a server at home. This could be a Raspberry Pi running home automation software that doesn't need a cloud server to function, but does all its processing on-device (or on the edge, as it's called now). You can perfectly do this with Node-RED or Home Assistant (see Chapter 10). Then there's no way a company can render your home automation system useless by shutting down their service, or your bathroom light doesn't turn on at night because your internet

connection or IFTTT's servers are down. You're fully in control of your home automation system.

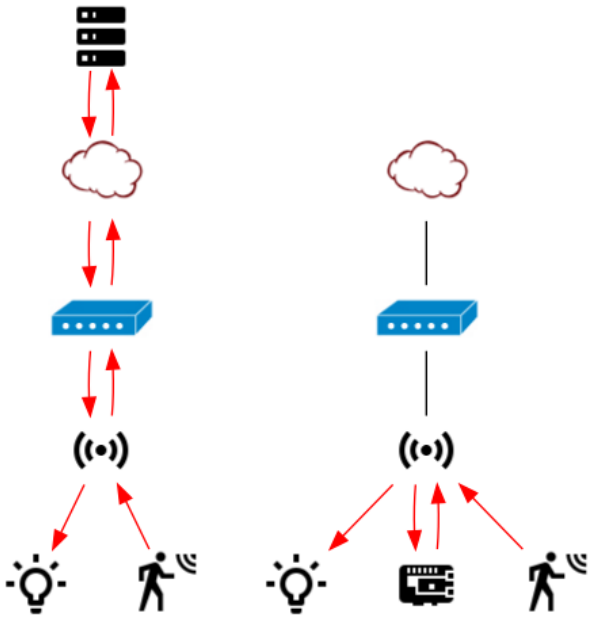

Figure 1.3 The home automation at the left is cloud-based: a simple motion detection message first goes to a server over the internet before returning to your light. The self-hosted system on the right makes much more sense: a Raspberry Pi on your network relays the message without using the internet detour.

Using cloud services for your home automation system has another risk: it invades your privacy. Look at some of the privacy issues with the services I talked about above:

- If you use the Ring video doorbell, it sends a video of everyone that steps on your porch to the manufacturer. The Ring company (which has been bought by Amazon in 2018) has a questionable approach to privacy: in January 2019 it was uncovered that employees have access to the video recordings of all Ring devices and even that the data are stored unencrypted.
- If you use the Nest Learning Thermostat, Google knows precisely when you are home and when you aren't.
- If you run a smart speaker like the Amazon Echo, what you tell your house members will be sent countless times inadvertently to Amazon because the Echo thinks it has heard its wake word.[7] Moreover, Amazon's employees listen to a part of all 'conversations' with the Echo to improve its algorithms.
- If you use the IFTTT service to link your various other services, you give one company

7 In 2020, a team of researchers at Northeastern University and Imperial College London simulated real-world conditions by playing popular TV shows. They found that a variety of smart speakers would activate by mistake up to 19 times each day on average. The study can be found here: https://moniotrlab.ccis.neu.edu/smart-speakers-study/.

access to all your home automation services, which is too much power concentrated in one company's hands: they can see exactly what you're doing.

1.4 ● How to use this book

This book describes a lot of components to support various home automation and IoT protocols. I don't expect that everyone will want to support all these protocols by installing these components manually. Home automation platforms such as Home Assistant (described in Chapter 10) have a fairly complete support for these and many more protocols. You can use such a platform to turn your Raspberry Pi into a home automation gateway.

However, I'm also a big believer in choice. You will have your preferences. Perhaps:

- you don't like these user-friendly home automation platforms.
- you do like one of these platforms, but it doesn't support one of the protocols.
- one of these platforms does support the protocol, but only with a local transceiver and you need a transceiver in another location for better coverage.

In all of these cases, you can use one of the open-source projects in this book to link this protocol to your gateway.

You can look at this book as a DIY manual to create your home automation gateway from scratch. But it's also describing a software architecture for a secure, modular, open-source, and self-sufficient home automation system. It's up to you to choose the components that implement this architecture in your own house, and if a home automation platform such as Home Assistant or Mozilla IoT WebThings Gateway does the job for you, that's fine. They don't lock you into their way of doing things, so you could perfectly use them and still link them to other systems thanks to MQTT and other protocols.

This book expects some familiarity with Raspberry Pi OS (formerly called Raspbian) or Linux in general. I explain most commands the first time I use them, but if you have never worked with a Linux system, I recommend reading an introductory text about Linux, Debian, Raspberry Pi OS, or bash (the Linux command line used in this book).

In various chapters, I show short Python programs that interact with your home automation system. The Python code in this book is not that advanced. If you don't know Python, you should be able to understand what the code does, and maybe you will even be able to adapt it because Python is known for its clarity. However, if you want to make the most out of this book, I do recommend you to learn some Python. The official Python documentation (https://docs.python.org/3/), especially the Python tutorial (https://docs.python.org/3/tutorial/index.html), is a good way to start. A home automation system is typically something very personal, and being able to program it is the best way to customize[8] it to your taste.

8 All Python code in this book has been developed for and tested on Python 3. Its predecessor, Python 2, has been retired on January 1 2020 (https://pythonclock.org), and shouldn’t be used anymore.

Note:
This book is not about how you connect sensors, relays, and so on directly to your Raspberry Pi using the GPIO header, but it strictly covers how you use your Raspberry Pi as a home automation gateway, collecting data and controlling devices remotely using radio and network protocols. Of course, it's perfectly possible to let your Raspberry Pi combine both tasks, but I wouldn't recommend it because it could lead to stability problems, and your gateway should be as reliable as possible.

Here's a short overview of what I'll cover in this book:

Chapter 1: Introduction
The theoretical foundation for this book, with my vision of what good home automation should look like and why you should use a Raspberry Pi for it.

Chapter 2: The Raspberry Pi as a home automation gateway
The practical foundation for this book, where you prepare your Raspberry Pi for its task as a home automation gateway.

Chapter 3: Secure your home automation system
Some general computer security principles and specific instructions to keep your home automation gateway secure, including encryption of all network traffic.

Chapter 4: MQTT (Message Queuing Telemetry Transport)
The lightweight network protocol that is at the center of this book, including the installation of an MQTT broker with encryption and authentication of all messages and the exploration of some MQTT clients.

Chapter 5: TCP/IP
Some TCP/IP-based protocols that everyone can use for home automation without the need for specialized transceivers, such as Wake-on-LAN, SSH, SNMP, HTTP/REST, and video surveillance systems.

Chapter 6: Bluetooth
An introduction to Bluetooth Low Energy, including a way to investigate Bluetooth Low Energy devices and using them to read sensor data and for presence detection.

Chapter 7: 433.92 MHz
The use of the RTL-SDR dongle to receive sensor measurements from devices transmitting on the popular 433.92 MHz frequency.

Chapter 8: Z-Wave
An introduction to the Z-Wave mesh protocol for wireless home automation.

Chapter 9: Zigbee
An introduction to the Zigbee mesh protocol for wireless home automation made popular

by Philips Hue and IKEA TRÅDFRI products.

Chapter 10: Automating your home
Complete home automation platforms such as Node-RED, Home Assistant, and AppDaemon, including dashboards for your home automation gateway.

Chapter 11: Notifications
Email and push notifications to warn you about events in your home, including an easy way to create notifications on receiving specific MQTT messages.

Chapter 12: Voice control
Voice control for your home automation gateway, without depending on online servers.

Chapter 13: Remote access
An overview of ways to remotely access your home automation gateway, with specific instructions about how to create a VPN.

Chapter 14: Conclusion
A wrap-up of this book, with a dashboard for all the discussed services.

Appendix
Some specialized tips that could come in handy in various situations.

Note:
Code examples from this book are published on https://github.com/koenvervloesem/raspberry-pi-home-automation. They can be copied from the GitHub repository one-by-one when trying the various applications in this book. You can also download them all at once. Refer to the instructions in the GitHub repository for more information.

1.5 ● Summary and further exploration

In this introductory chapter, I gave a theoretical foundation for this book, with my vision of what good home automation should look like and why you should use a Raspberry Pi for it. I argued why your home automation system should be:

- secure
- modular
- open-source
- self-sufficient

I hope the examples I gave you have convinced you that these are important properties of your home automation system. In the rest of this book, I show you how you create such a system with a Raspberry Pi.

If you want to read more about some issues I raised in this chapter, I can recommend

the special edition of Mozilla's Internet Health Report of November 2019, called "*Privacy Included: Rethinking the Smart Home" (https://foundation.mozilla.org/en/privacy-included/). It talks about the so-called "smart home" from a more general point of view and is much in line with the approach I advocate in this book. The report stresses the importance of privacy, security, interoperability, and sustainability for smart home devices.

Note:
The use of the ⏎ symbol denotes that code should be typed contiguously on the same line.

Chapter 2 ● The Raspberry Pi as a home automation gateway

While the previous chapter gave you a theoretical foundation with an overview of what you're going to build in this book (and especially why), the second chapter gives you a practical foundation: it's all about preparing your Raspberry Pi for its task as a home automation gateway.

I'll first talk a bit about which Raspberry Pi models are suitable for home automation, after which you'll learn how to install Raspberry Pi OS, configure network connectivity and access your Raspberry Pi over the network using SSH. I also talk a bit about the tmux terminal multiplexer, which is indispensable for many tasks in this book, and about installing Python packages. I'll finish this chapter by installing Docker, which is essential to be able to install various programs in later chapters without messing up your base system.

2.1 ● Which Raspberry Pi models are suitable for home automation?

Since its first release at the beginning of 2012, various Raspberry Pi models have been released. Most of them, even the first ones, are suitable for home automation, but not every model is up to every task. I have been running the Domoticz home automation software on the first Raspberry Pi B model for more than five years to control my home, and it has only been replaced by a newer model in the last three years.

So if you have an older model like the Raspberry Pi 2 or even the first model lying in your cupboard, by all means, try it. Whether you can use it as your home automation gateway depends on how many devices you want to control and which extra software packages you want to run on your Raspberry Pi.

Note:
These early models don't have Wi-Fi or Bluetooth. If you want to use these wireless protocols, you'll have to use adapters that plug into the USB ports.

In most cases, I recommend you to use a Raspberry Pi 3B or 3B+ as your home automation gateway. These models have enough processing power to run most home automation workloads, and they have Ethernet, Wi-Fi, and Bluetooth. The Raspberry Pi 3A+ (which has the same processing power as the 3B+) is fine if you don't need Ethernet connectivity and one USB 2.0 port is enough for you. You can always connect more USB devices with a USB hub. But note that 512 MB RAM could be insufficient for some applications.

Figure 2.1 In many home automation scenarios the Raspberry Pi 3B+ is up to the task.

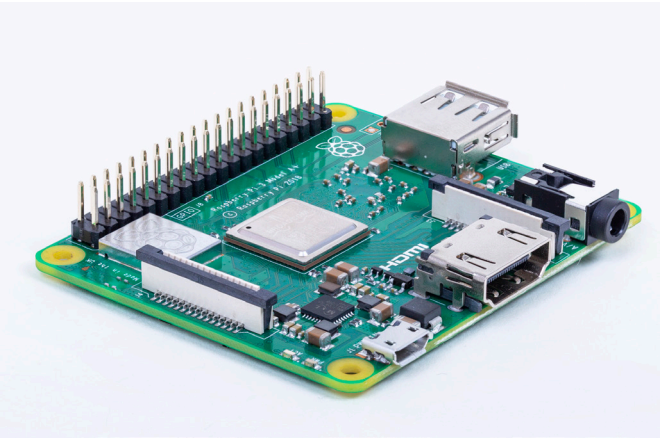

Figure 2.2 The Raspberry Pi 3A+ is fine as a home automation gateway if you don't need Ethernet connectivity and one USB 2.0 port is enough.

Only if you're serious about home automation and are running very demanding workloads, or need the USB 3.0 and Gigabit Ethernet throughput, would I recommend choosing a Raspberry Pi 4 as your home automation gateway.[1] This model exists in versions with 1, 2, 4 and 8 GB. Choose the RAM size depending on your workload.[2]

1 The Raspberry Pi 4 is essential if you're running AI (artificial intelligence) tasks such as image recognition with the Google Coral USB Accelerator: this TensorFlow Light accelerator dongle needs USB 3 acess to use its full speed.

2 The model with 1 GB RAM has been retired in the beginning of 2020, but if you have still one of these, it's perfectly usable for most home automation tasks.

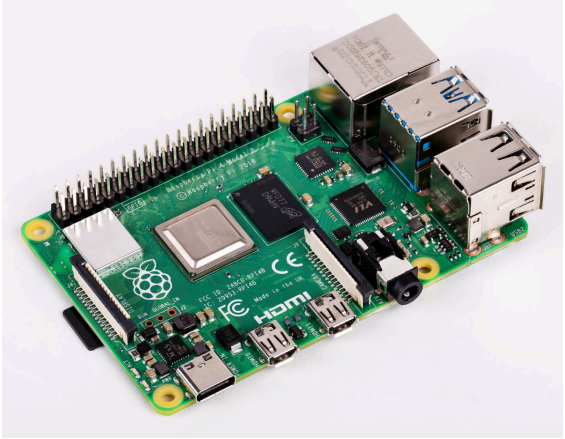

Figure 2.3 You only need the Raspberry Pi 4 for very demanding home automation workloads, for instance for image recognition.

The Raspberry Pi Zero W(H) has the same processor as the first Raspberry Pi model, but clocked at 1 GHz instead of 700 MHz. So this is only suitable for minimal home automation workloads. It has Wi-Fi and Bluetooth connectivity, but no Ethernet and only one micro-USB OTG connector. So for a gateway, the Raspberry Pi Zero W(H) is seriously handicapped. On the other hand, because of its low price, small size, and low power consumption, it's a very interesting model to read Bluetooth sensors at various places in your house, for instance, to detect in which room you are (see Chapter 6). And in combination with a Raspberry Pi Camera the Raspberry Pi Zero W can function as a cheap IP camera.

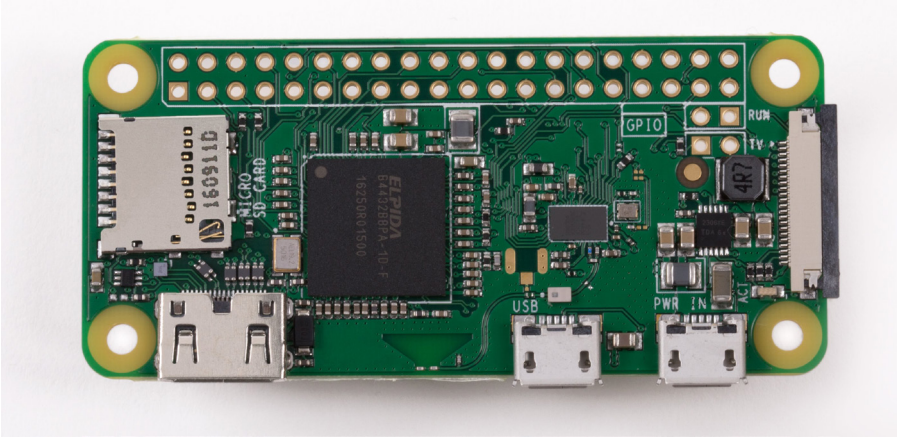

Figure 2.4 The Raspberry Pi Zero W(H) is not powerful enough to run a serious home automation gateway, but it's perfect as a simple gateway for Bluetooth or Wi-Fi sensors or other devices.

There are still other Raspberry Pi models: the Compute Module, Compute Model 3, and 3+. These are meant for industrial use. While they have their advantages, I'm not going to use them in this book. This is, after all, a book about home automation, not industrial

automation. But if by any chance you have one of these, feel free to try them: they should work because they are fully compatible with software for the 'normal' Raspberry Pi models. To sum up, this table shows some specifications of the Raspberry Pi models that are most relevant for home automation:

	Pi 4B	Pi 3B+	Pi 3A+	Pi 3B	Pi Zero W(H)
SoC	BCM2711	BCM2837B0	BCM2837B0	BCM2837	BCM2835
Cores	4	4	4	4	1
CPU clock	1.5 GHz	1.4 GHz	1.4 GHz	1.2 GHz	1.0 GHz
RAM	1 / 2 / 4 / 8 GB	1 GB	512 MB	1 GB	512 MB
USB	2 × USB 3.0, 2 × USB 2.0	4 × USB 2.0	1 × USB 2.0	4 × USB 2.0	1 × micro-USB 2.0 OTG
Ethernet	Gigabit	Gigabit over USB 2.0	/	10/100 Mbps	/
Wi-Fi	802.11b/g/n/ac dual-band	802.11b/g/n/ac dual-band	802.11b/g/n/ac dual-band	802.11n	802.11n
Bluetooth	5.0	4.2	4.2	4.1	4.1
Size	85.6 mm × 56.5 mm	85.6 mm × 56.5 mm	65.0 mm × 56.5 mm	85.6 mm × 56.5 mm	65.0 mm × 30.0 mm

Table 2.1 Raspberry Pi models for home automation

2.2 • Requirements for a reliable home automation gateway

The Raspberry Pi is an easy computer board to experiment with. If you pick a Raspberry Pi, put a Raspberry Pi OS image on the microSD card, and start installing some programs explained in this book, you'll have a home automation gateway in no time.

It's tempting to start this way, and if it's just to see what the possibilities are, that's no problem. But you'll run your house on this Raspberry Pi, so you'll have to at least think about how to make it reliable. These are some aspects you have to consider:

Place

Your home automation gateway should be in an accessible place so it's easy to troubleshoot it or attach or replace adapters. But at the same time, you don't want it to be too accessible. For instance, if you have it lying on your desk, the risk is too high that you spill coffee on it, yank off its power cable, or just that it's in your way.

Storage

The capacity of the microSD card for your Raspberry Pi should be high enough to store all software, logs, and data. If you install all projects in this book, you need a microSD card of at least 16 GB. I recommend 32 GB to be on the safe side. Because microSD cards are notoriously unreliable, especially after a power failure, regularly take a backup of the most

important data.[3]

Case
It's perfectly acceptable to use the Raspberry Pi without a case when you are experimenting. But as soon as you're starting to rely on it as a home automation gateway, you should put it in a case: this protects your Raspberry Pi from dust or small accidents. Choose a case fit for your purpose. For instance, if you need to use the GPIO pins, make sure the case gives access to them.

Cooling
Especially with the Raspberry Pi 4, the temperature mustn't rise too much. If the temperature of the SoC is between 80 and 85 degrees Celsius, the ARM core(s) will be throttled back in an attempt to reduce the core temperature. If the temperature of the SoC reaches 85 degrees Celsius, the GPU will also be throttled back. Running for too long on these high temperatures is detrimental to the health of the processor. If you are doing intensive computations on your home automation gateway, add a heat sink or a fan. There are even cases that double as a heat sink with their body.

Power
Use a reliable power supply for your Raspberry Pi, as you don't want your home automation gateway to fail when the power requirements are not met temporarily. The official Raspberry Pi power supply is a safe choice, but other adapters that supply 5.1 V 2.5 A (or 3 A for the Raspberry Pi 4 to be on the safe side) should be fine too. Note that the power consumption depends on the peripherals you have connected, such as USB transceivers or the Raspberry Pi Camera Module (which requires 250 mA).

Network
Wi-Fi connections are not as reliable as Ethernet connections. If it's possible, use an Ethernet connection for your home automation gateway. This way your house doesn't lose its head when your Wi-Fi access point fails.

3 In this book, you only need to copy the docker-compose.yml file and the contents of the directory /home/pi/containers to have all your home automation data.

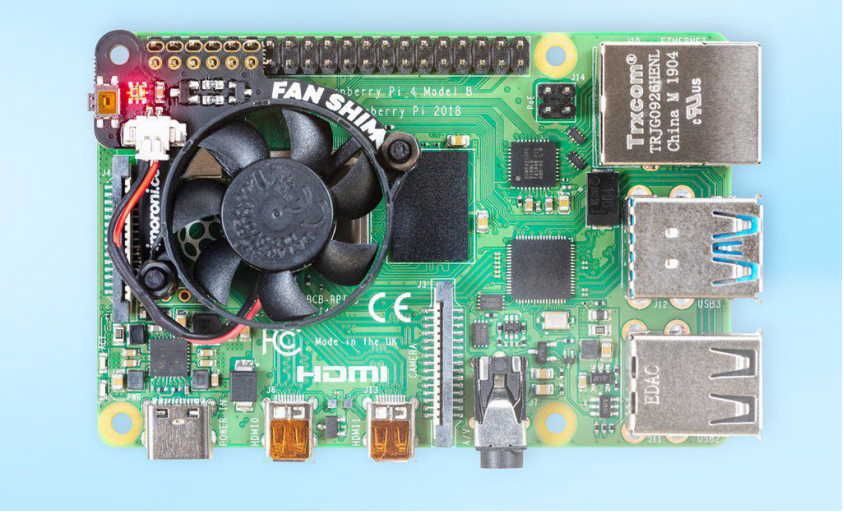

Figure 2.5 Pimoroni's Fan SHIM is an efficient cooling solution for the Raspberry Pi 4.

Note:
If you have a power outage in your home, the microSD card of your Raspberry Pi may become corrupt and your home automation gateway will not work anymore after the power comes back on. If you want to protect for this scenario, use a UPS (uninterruptible power supply) for your Raspberry Pi. It has an emergency battery with enough power to safely turn off your Raspberry Pi.

2.3 • Installing Raspberry Pi OS

In this book, you'll use Raspberry Pi OS (formerly called Raspbian) as the operating system for your Raspberry Pi. Raspberry Pi OS is a Linux distribution based on Debian GNU/Linux, optimized for its use on a Raspberry Pi. It's the official operating system that the Raspberry Pi Foundation recommends.

If you visit the download page on https://www.raspberrypi.org/downloads/raspberry-pi-os/, at the moment I'm writing this book you'll see three variants of Raspberry Pi OS:

- Raspberry Pi OS (32-bit) with desktop and recommended software
- Raspberry Pi OS (32-bit) with desktop
- Raspberry Pi OS (32-bit) Lite

You'll use the Lite version: this doesn't have a graphical desktop, so you have to work with Raspberry Pi OS using the command line. Some familiarity with the Linux command line is therefore recommended, but I'll explain the commands that you need.

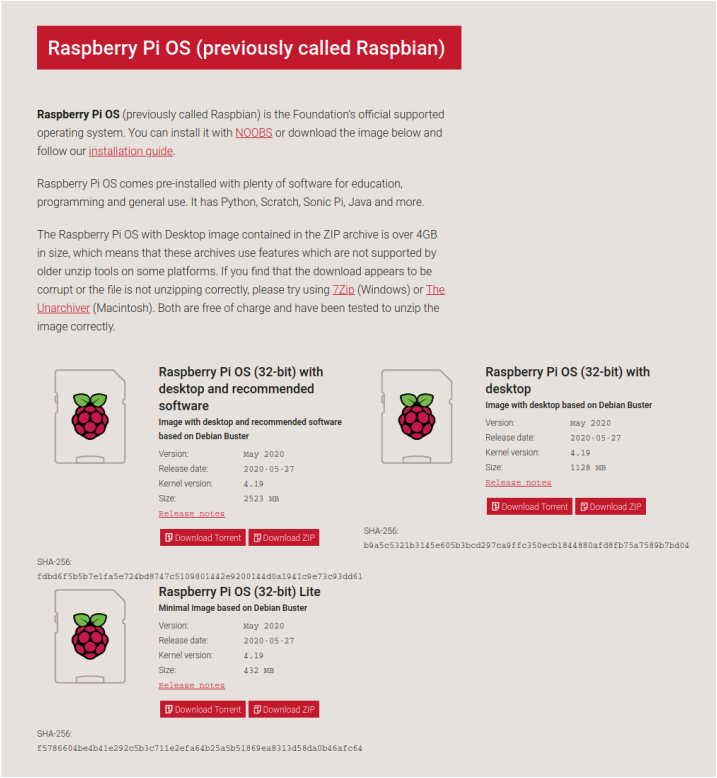

Figure 2.6 In this book you'll use Raspberry Pi OS Buster Lite as the base operating system for your home automation gateway.

I'm choosing the Lite version for several reasons:

- The less software is installed, the more secure the setup (see Chapter 3).
- The less software is installed, the faster the operating system runs.
- Software for a home automation gateway doesn't need the graphical desktop, because it has its own user interface.

So, download the Raspberry Pi OS Buster Lite image (click on **Download ZIP**), and then download balenaEtcher from https://www.balena.io/etcher/. This program is available on Windows, macOS, and Linux and offers an easy way to write operating system images to (micro)SD cards.

Now put a microSD card in your computer's card reader. If your computer hasn't one, you'll need an external card reader that you connect to a USB slot of your computer.

If you start balenaEtcher now, you just have to choose the image and the drive letter of the microSD card you want the image to be written to. Make sure that you have chosen the right image and drive letter, and then click **Flash!** to begin writing data to the microSD card.

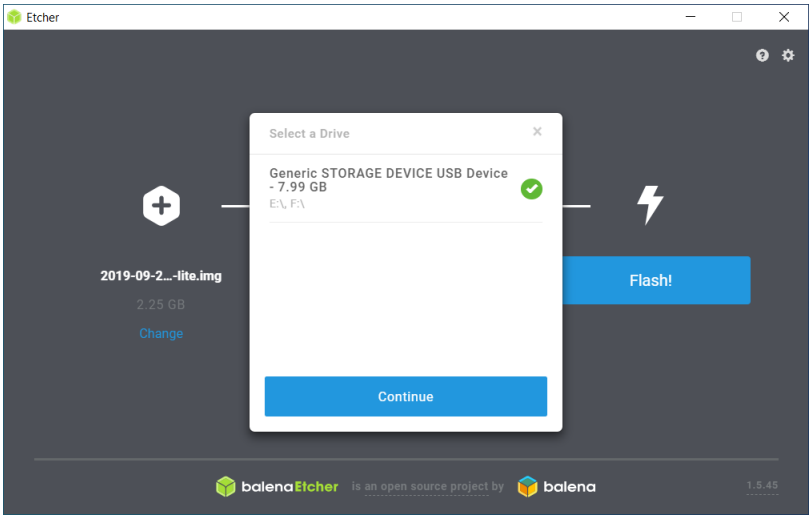

Figure 2.7 Make sure that you choose the correct drive to write the Raspberry Pi OS image to.

Note:
You don't have to unzip the zip file: balenaEtcher automatically unzips it before writing the img file in it to the microSD card.

Warning:
Double-check whether you chose the correct drive letter before you write it to your microSD card: if you accidentally select the wrong drive, all data is overwritten!

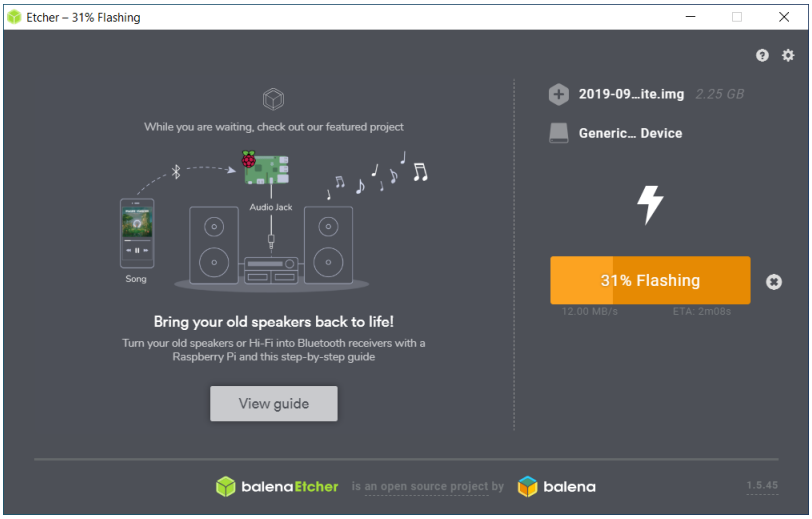

Figure 2.8 Write the Raspberry Pi OS image to a microSD card with balenaEtcher

For more advanced options, consult the Raspberry Pi Foundation's documentation about installing operating system images (https://www.raspberrypi.org/documentation/installation/installing-images/).

When you have successfully written the microSD card, remove it from the card reader but don't put in your Raspberry Pi's microSD card slot yet: reinsert it in your computer's card slot. A boot volume should be mounted.

2.4 ● Setting up network connectivity with Ethernet or Wi-Fi

As you are not using a desktop operating system on the Raspberry Pi, you have to configure it fully on the command line. You could now connect a keyboard and monitor to the Raspberry Pi and configure it like this, which is the way shown in many other tutorials. But you won't need the keyboard and monitor later to run a home automation gateway, so it doesn't make much sense to use them now. Running a computer without a keyboard and monitor is called a headless system.

But without a keyboard and monitor to type commands and see some output, you need another means to interact with the Pi: the network. So first you need to set up network connectivity on your Pi.

2.4.1 ● Ethernet

If you connect your Raspberry Pi to your home network's router using an Ethernet cable, the connectivity should become configured automatically: Raspberry Pi OS asks for an IP address to your network's DHCP server (usually running on your router), and this returns an IP address that the Pi can use on your network. It's this IP address that you need to know to connect to your Pi.

But what is this IP address? Raspberry Pi OS shows this on the login screen when you have attached a monitor, but in headless mode, there's another way to discover your Pi's IP address: visit the web interface of your router and search for the DHCP leases of your DHCP server. There you should see your Raspberry Pi with its MAC and IP address. The MAC address of every Raspberry Pi starts with b8:27:eb (the older models) or dc:a6:32 (beginning from the Raspberry Pi 4), so it's easy to find. If you have more than one Raspberry Pi on your network, you'll have to guess which one is your home automation gateway…

I recommend Ethernet for your home automation gateway because it's more reliable than Wi-Fi: this way your home automation gateway doesn't depend on your Wi-Fi access point working correctly and Wi-Fi signals coming through.

Note:
Especially when you depend on your home automation gateway to secure your home, for instance with smart locks or security cameras, ethernet is much more reliable and secure: Wi-Fi signals can be blocked with a Wi-Fi jammer in your environment.

2.4.2 ● Wi-Fi

Although I generally don't recommend using Wi-Fi for your home automation gateway, there are situations where you don't have any other choice, for instance, if your gateway is in a place where you don't have an Ethernet socket. Of course, you can use Wi-Fi in this instance.

When you reinserted the microSD card into your computer's card slot, you should have seen a boot volume being mounted. Now create a file with the name wpa_supplicant.conf in this folder, with the following configuration:

```
country=COUNTRYCODE
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
  ssid="SSID"
  psk="PASSWORD"
}
```

Instead of `COUNTRYCODE`, `SSID` and `PASSWORD`, enter your country's ISO 3166-1 alpha-2 code (BE for Belgium, NL for the Netherlands, DE for Germany, and so on)[4] and the SSID (Service Set Identifier) and password of your Wi-Fi access point.

Then save this configuration file, and be sure to save it as plain text, not as a Word file or any other rich text format.

2.4.3 ● Setting a fixed IP address

Your Raspberry Pi will get an IP address from the DHCP server of your router, and this will probably stay the same. This is important because you will use the IP address of your Raspberry Pi to access all services running on your home automation gateway.
You should make sure that the IP address of your home automation gateway always stays the same. One way to do this is in the DHCP settings of your router. The exact procedure depends on your model, but it should be called something like "DHCP static mappings".

Add a mapping where you enter the Raspberry Pi's MAC address and the corresponding IP address that you want to assign to it.

4 See https://en.wikipedia.org/wiki/ISO_3166-1 for the full list of ISO-3166-1 country codes.

Note:
If you assign another IP address in the DHCP static mappings to the one your Raspberry Pi currently has, reboot your Raspberry Pi to be sure that it is assigned the correct one.

2.5 ● Remote access using SSH

Whether you choose Ethernet or Wi-Fi, your Raspberry Pi should now have access to your network when you start it from this microSD card. But you still need something else: you should have access to the Raspberry Pi over the network. You'll get this access using SSH (Secure Shell), which is the default remote login solution for Linux and UNIX computers.

In this section, you'll make your Raspberry Pi accessible in your home network with OpenSSH. After this, all commands that I show in this book can be entered remotely. So you can configure your home automation gateway from everywhere in your home: from your desktop PC, your laptop, even your tablet or smartphone if you have installed an OpenSSH client app on it.[5]

2.5.1 ● Enabling the SSH server

Raspberry Pi OS already has an SSH server, but it doesn't run by default because it would open a security hole: everyone on your network who knows the default password for Raspberry Pi OS, would be able to log in. To enable SSH, just create an empty file called ssh in the boot folder of the microSD card.

Then unmount the microSD card, put it into the microSD card slot of your Raspberry Pi, (optionally) connect the Ethernet cable between your Raspberry Pi and your router or switch, and finally connect the power adapter to your Raspberry Pi, after which it boots. Now have a bit of patience: Raspberry Pi OS is booting, and the first time this takes a while because it expands the file system to use the microSD card's full capacity. In the meantime, have a look at the DHCP leases of your router to discover your Raspberry Pi's IP address.

2.5.2 ● Connecting with the SSH client

Now you'll use the OpenSSH client to log into the Raspberry Pi over the network. If you're running a recent Windows 10 release, the OpenSSH client is automatically installed. On Linux and macOS, it's also generally installed. Just open a command prompt and enter:

```
ssh pi@IPADDRESS
```

Substitute your Raspberry Pi's IP address for `IPADDRESS`. The `pi` before the `@` sign is the

5 You could also connect a keyboard and a display to your Raspberry Pi, log in and enter all commands this way, but then you'll have to physically sit at your Raspberry Pi every time you want to install or configure something on your home automation gateway.

default user in Raspberry Pi OS.

If the Windows command prompt doesn't understand the `ssh` command, maybe you still have to install the OpenSSH client. To do this, open **Settings**, then go to **Apps > Apps and Features > Manage Optional Features**. At the top of the window select **Add a feature**, then locate **OpenSSH Client** and click **Install**. After this, you can use the `ssh` command in the command prompt.

Note:
If you can't find the OpenSSH client in the optional features of the Windows settings, you're running an older Windows version. You should then follow the instructions of Microsoft's project Win32-OpenSSH.
(https://github.com/PowerShell/Win32-OpenSSH/wiki/Install-Win32-OpenSSH)
or download PuTTY
(https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html). The latter has a graphical interface.

Now when you first log in to your Raspberry Pi with the `ssh` command, it will show you an SSH fingerprint. Accept it by entering **yes** and pressing **Enter**, and after this enter the password **raspberry**.[6]

If you entered the correct password, you see something like this in your terminal:

```
The authenticity of host '192.168.0.178 (192.168.0.178)' can't be established.
ECDSA key fingerprint is SHA256:64q3h9q5eoGZxvoLqueCEjFFF0PM6BIhvv6KPDkuvb4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.178' (ECDSA) to the list of known hosts.
pi@192.168.0.178's password:
Linux raspberrypi 4.19.66-v7+ #1253 SMP Thu Aug 15 11:49:46 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 28 07:19:24 2019 from 192.168.0.115
pi@raspberrypi:~ $
```

Figure 2.9 You use the ssh command to log in to your Raspberry Pi remotely.

The last line, `pi@raspberrypy:~ $`, is the command prompt. The `pi` is your username, `raspberrypi` is the default hostname of your Raspberry Pi (you'll change this in the next

6 To be sure that no one is executing a man-in-the-middle (MITM) attack on you and presenting you a rogue SSH server instead of your Raspberry Pi, you should check whether the shown SSH fingerprint matches the fingerprint of the actual SSH host key of your Raspberry Pi. If you want to be sure, log into your Raspberry Pi with a keyboard and monitor, enter the command `ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key.pub` and verify whether the output matches the fingerprint shown to you by the `ssh` command.

section) and the ~ is your current working directory, which is a shortcut to your home directory, `/home/pi`. After the dollar sign, you can start typing commands.

What you see now is called the `bash` shell. It's like the command prompt in Windows. You type commands, and these commands show some output.

2.6 ● Basic setup

As soon as you're logged in, change the default password of the pi user with the following command:

```
passwd
```

Choose a strong password: at least 12 characters long, and no one else should be able to guess it.

Warning:
Choosing a strong password for a Raspberry Pi in your home network may sound paranoid, but it's really important. If by any chance you make an error and expose your Raspberry Pi to the internet, anyone can log in to your Raspberry Pi when they guess your password, and they can use your Raspberry Pi as a stepping stone for breaking into other systems in your home. See Chapter 3 for more information about securing your Raspberry Pi and choosing a strong password.

Then update the package repositories and upgrade all packages to their newest version:

```
sudo apt update
sudo apt upgrade
```

If you get the question **Do you want to continue? [Y/N]**, confirm with the Enter key. Now all newest versions of the installed software packages are downloaded and installed.

After this, configure some extra settings by running the Raspberry Pi OS configuration program:

```
sudo raspi-config
```

This opens the Raspberry Pi Software Configuration Tool, which offers a command-line

menu-based way to configure basic settings. You can go through the menu with the arrow keys, select a menu with Enter, and go to the 'buttons' below with the Tab key. Now change the following essential settings:

- **2 Network Options - N1 Hostname**: Change the hostname of your Raspberry Pi to something else than **raspberrypi** if you have more than one Raspberry Pi running at home.
- **4 Localisation Options - I2 Change Timezone**: Choose your timezone to get the right time.
- **7 Advanced Options - A3 Memory Split**: Change the RAM that the GPU gets to the minimal value, **16 MB**. As your Raspberry Pi operates as a home automation gateway and doesn't need a graphical interface, giving more RAM to the GPU is just a waste of memory.

After this, go to **Finish** in the main menu and press Enter. If you're asked to reboot now, confirm with **Yes**.

2.7 ● The tmux terminal multiplexer

Suppose that you're logged into your Raspberry Pi using SSH and you are running a `sudo apt upgrade` command. It's a long upgrade, and you want to do something else in the meantime. Should you log into your Raspberry Pi again then using SSH?

Fortunately not. Because you're going to work a lot on the command line in this book and you'll regularly have to do various tasks at the same time, I'll show you a tool that will help you work more efficiently: a terminal multiplexer. This lets you run multiple terminal sessions in one window and adds various other interesting features, such as shortcuts to copy and paste text.

In this book, I'm working with `tmux` (https://tmux.github.io). Install it like this:[7]

```
sudo apt install tmux
```

2.7.1 ● The basics of tmux: windows

Now just run `tmux`. It looks like you're in just another terminal session, but there's one difference: there's a status bar visible on the bottom. To the right shows the Pi's hostname and current date and time.

When you start a tmux session, this opens one window, where you can start typing Linux commands. With Ctrl+b followed by c (that is, push Ctrl and b simultaneously, release both keys, and then push c only) you open a new window. It's like you started a completely new

7 Another well-known terminal multiplexer is screen (https://www.gnu.org/software/screen/).

terminal window, for instance from a separate SSH session.

The status bar shows you all windows with an index number and the name of the currently active program. For now, this is `bash`, the shell that lets you type Linux commands. Always keep an eye on the names of the tmux windows, because it can give you interesting information. For instance, if a long upgrade command is running in one window and you're running commands in another window, and you suddenly see that the name of the first window changes to `bash`, you know that the upgrade process is complete.

Moving from one window to the next is easy: you use Ctrl+b n (for next) or Ctrl+b p (for previous). Ctrl+b followed by a number moves you to the window with that specific index number, for instance, Ctrl+b 0 for the first window.

You can exit a tmux window like any other terminal session, with `exit` or the Ctrl+d key combination. If there's only one window, exiting the window exits `tmux` too, and you return to the terminal session where you started the tmux command. If at any time the program in a `tmux` window 'hangs', you can exit the window with Ctrl+b &.

2.7.2 ● Working with tmux sessions

All programs running in your terminal since you started tmux are in a 'session'. The cool part about tmux is that you can 'detach' this session, with Ctrl+b d. A detached session keeps running in the background as long as your Raspberry Pi is on, even after you have logged out.

You can also reattach to the session:

```
tmux attach
```

After this, all your tmux windows are visible again and you can interact with them like before. This is helpful if you have started a long upgrade process on your Raspberry Pi but then the computer from which you have logged into your Raspberry Pi using SSH has to reboot. If you had just started the upgrade process directly from the shell, the upgrade process would stop as soon as your session ended because the network connection was disrupted. With tmux, you just detach the session, log off and let the process run while you reboot your computer and log on again.

Note:
You can create multiple sessions and even give them names. See `man tmux` for this and much more.

2.7.3 ● Seeing more at the same time with panes

Another nice feature of tmux are panes: they let you create multiple terminal sessions next to each other in the same window. This is useful if you're editing a configuration file in one pane, running commands in another pane, and looking at the output of a log file with `tail -f` in a third pane. You see al these panes at a glance without having to move from one window to another one.

By default, tmux starts one pane in each window. Hitting Ctrl+b % divides this pane into two panes next to each other and brings you to the new pane. Ctrl+b " divides the current pane into two panes above each other. With Ctrl+b o you jump from one pane to the next. With Ctrl+b and an arrow key, you jump to the next pane in the direction of the arrow.

Note:
Ctrl+b followed by the space bar cycles through all default layouts of the panes: from left to right, from top to bottom, one big pane at the top and the other ones above, one big pane at the left and the other ones at the right, or all panes equally sized.

Panes are not 'stuck' in their window indefinitely. Sometimes you have created a couple of panes in a window and you want to open one of these panes in its own window. This is very easy to do: hit Ctrl+b ! and tmux moves the current pane to its own window. If you just want to temporarily show one pane fullscreen, hit Ctrl+b z. The same keyboard combination brings you back to the normal layout. Exiting a pane is done the same as exiting a window: with the `exit` command or Ctrl+d.

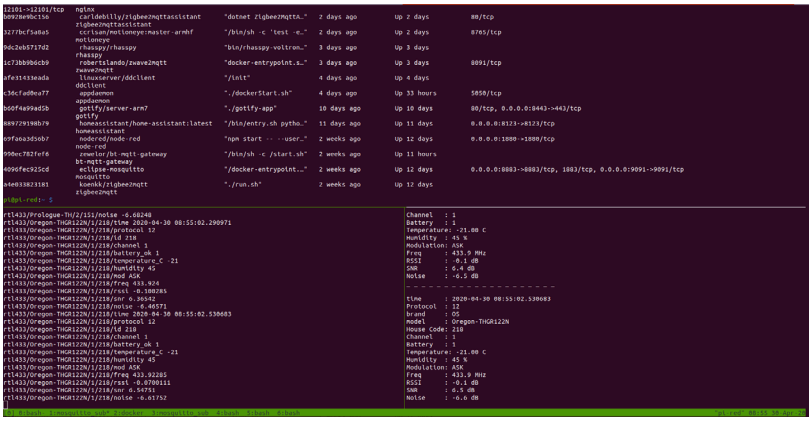

Figure 2.10 With tmux panes and windows you can execute a lot of terminal sessions without losing the overview.

2.7.4 ● Copying and pasting text

Terminal sessions are all about text: the content of a configuration file or log file, the output of a command, … In the very powerful copy mode of tmux, you can copy and paste text from your terminal session. You enter the copy mode with Ctrl+b [.
As soon as you enter the copy mode, you see two numbers between square brackets at the top right of your current pane. These numbers are the current and maximum line numbers of all output that has been written since the beginning of the current pane's session.
In the copy mode, you can navigate through this output with the arrow keys. This way you can revisit output that has scrolled off your screen. Enter q to leave the copy mode.

The name of the copy mode comes from what you can do with it: copy text. But to enable this functionality, you first have to enable mode keys in the tmux configuration file:

```
echo "setw -g mode-keys vi" > ~/.tmux.conf
```

Exit tmux completely and then run tmux again.

Enter copy mode with Ctrl+b [. Move the cursor to the beginning of what you want to copy and press the space bar. Then move to the end of your selection and press Enter. All text between will be copied now. Now go to the place where you want to paste the text. This can be another panel or another window.[8] And then hit Ctrl+b] to paste the text. Or hit Ctrl+b = to get a list of all previously copied texts in this session and choose the one you want to paste.

2.8 ● Python

Many projects in this book use the Python programming language. Python 3 is installed by default in Raspberry Pi OS, but there exist a lot of external packages, and many of them are available in the Python Package Index (PyPI), which can be found on https://pypi.org. These additional Python packages from PyPI can be installed with the `pip` package manager.

So first install it with:

```
sudo apt install python3-pip
```

8 You can even copy and paste between different tmux sessions, if you are working with multiple sessions.

Note:
Always use `python3`, `pip3` (and so on) commands to use the Python 3 versions of these Python commands. If you forget to type the number 3, you are using the Python 2 versions, and Python 2 has been deprecated. Although in a Python 3 virtual environment the `python` command refers to `python3`, it's better to always use `python3` so you don't have to think about it.

Now you can install a Python package called 'foobar' globally with `sudo pip3 install foobar`, or only for the local user with `pip3 install --user foobar`. But I'm only going to do this a few times in this book, for some very specific packages.

The reason is simple: Python packages are in constant flux and if you install all the packages you need globally or locally there will be a moment where you bump in a problem where package A needs version 1.0 of packet Z and packet B needs version 2.0 of packet Z, which is incompatible with version 1.0. You can only install one version of a Python package this way, so package A or package B will stop working.

Note:
If you want to develop upon some of the Python projects used in this book, for instance you want to fix bugs or add support for extra devices, you'll have to download the source code. Most of the time this is done using the git command. Install it with `sudo apt install git`.

2.8.1 ● Virtual environments

Python has a solution for this 'dependency hell': virtual environments. A virtual environment is a directory where you install Python packages instead of in your global or user-local Python package directory. This virtual environment is isolated from other virtual environments and your global and user-local Python package directory.

The idea is that you create a new virtual environment for each non-trivial Python project that you want to install or develop. This way you're sure that you don't mess with the dependencies of other installed Python packages.

First, install the virtual environment functionality:

```
sudo apt install python3-venv
```

Creating a virtual environment in the directory `.venv`[9] is then easy:

9 This is a hidden directory because its name starts with a dot. This way commands like ls don't show the directory by default. Use `ls -a` to show hidden files and directories too.

```
python3 -m venv .venv
```

This creates the directory `.venv` and puts the Python 3 interpreter and the standard Python library in it, including the pip package manager.

If you want to use the Python interpreter from your virtual environment now, you have to 'activate' the environment:

```
source .venv/bin/activate
```

If you run `python3` after this, you don't use the system's Python command (in `/usr/bin/python3`), but the Python version in your virtual environment. You can check this with the command `which python3`. And if you install packages with `pip3`, they will be installed in the virtual environment. Leave the virtual environment with `deactivate`. Now the packages you installed are not available anymore.

Reactivating a virtual environment is just a matter of typing `source .venv/bin/activate` again in the right directory. You can have a virtual environment everywhere, and you can give it any other name than `.venv`. Because it's so easy to create a virtual environment, I recommend using virtual environments for almost all Python projects that have dependencies. It will save you some difficult to solve issues in the long run.

2.8.2 ● Package requirements

If you install a Python project with `pip3`, all its Python dependencies are installed automatically. But if you use a Python project that doesn't have a package and you have to download its source, it will probably have a list of its dependencies in a file called `requirements.txt`.

These dependencies are easy to install. Just create and activate a virtual environment in the directory of the project, and then install its dependencies with:

```
pip3 install -r requirements.txt
```

Some packages have non-Python requirements. Then you have to install these requirements with the Raspberry Pi OS package manager, `apt`. Note that these dependencies are not limited to the virtual environment then. Most of the time, this won't be problematic. But because eventually, you can encounter the same kind of dependency problems as with pip packages, a better idea is to use Docker. I'll explain this in the next section.

2.9 ● Docker

Docker (https://www.docker.com) is an easy way to run various services: it works by running each service in a separate container, which is isolated from all other containers. I use Docker a lot in this book.

Essentially each container you're running on Raspberry Pi OS using Docker is a minimal Linux system (an image) that makes use of the Raspberry Pi OS Linux kernel. If you're running a service in a Docker container, you can be sure that it doesn't pollute the Raspberry Pi OS. If you don't need a container anymore, you can just delete it without having to fear that you have done something wrong, forgot to delete some files, or deleted some essential files and your Raspberry Pi OS system doesn't work anymore.

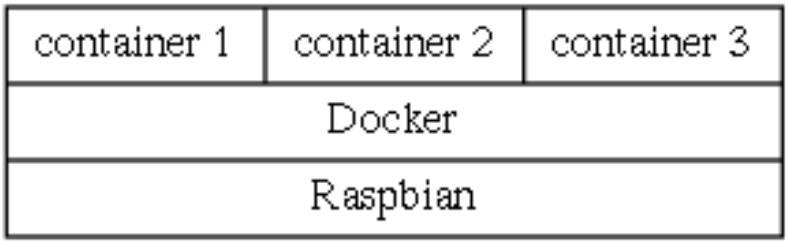

Figure 2.11 Docker isolates different programs from each other in containers.

2.9.1 ● Installing Docker

The easiest way to install Docker on Raspberry Pi OS is as follows:

```
curl -sSL https://get.docker.com | sh
```

Then give the `pi` user access to Docker:

```
sudo usermod pi -aG docker
```

Log out with `exit` and log in again, or reboot your Raspberry Pi with `sudo reboot`. Then execute the following command to see the status of the Docker engine:

```
systemctl status docker
```

If you're seeing **active (running)** in the output, Docker is running fine. To double-check your configuration, try running the hello-world container:

```
docker run --rm hello-world
```

If all goes well, you should see something like this:

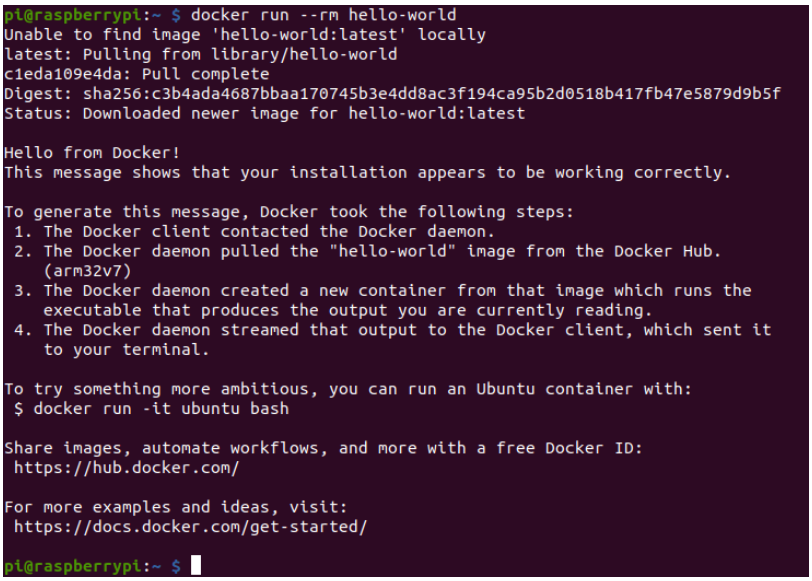

Figure 2.12 The hello-world Docker container runs fine on this system.

In the first line of Docker's output, you see the message "Unable to find image 'hello-world:latest' locally". Because the docker command can't find the image on your Raspberry Pi, it downloads it from the Docker Hub, a central repository with Docker images. After downloading the image, Docker creates a new container from that image and runs a program in that container showing you the helpful description from Figure 2.12. After this, the container is removed (because of the `--rm` option of the `docker` command).

Finally, create a directory where the data for your Docker containers will be stored:

```
mkdir /home/pi/containers
```

In this book, I'll use the convention that data for a container (configuration files, logs, data, and so on) will be put in a subdirectory of this `/home/pi/containers` directory, with the same name as the container.

Figure 2.13 After completing this book, the containers directory should look something like this.

2.9.2 ● Installing Docker Compose

After a few chapters you'll be running multiple Docker containers, and this can become difficult to maintain. That's why I'm using Docker Compose (https://docs.docker.com/compose) in this book. With Docker Compose you define all your containers in one YAML file.[10]

The advantage is that if you want to add, remove, or change services, you just have to modify this one configuration file. You can also create and start all your containers with a single command.

Docker Compose is distributed as a Python package, so you can install it with:

```
sudo pip3 install docker-compose
```

And verify whether it has been installed correctly by asking for the version number:

10 YAML is a recursive acronym for "YAML Ain't Markup Language".

```
docker-compose --version
```

2.9.3 ● Creating a Docker Compose YAML file

As an example of the Docker Compose YAML file, and to double-check whether Docker Compose works, create a file `docker-compose.yml` in your user's home directory:

```
nano docker-compose.yml
```

This opens `nano`, the Raspberry Pi OS default command line text editor, and creates an empty file called `docker-compose.yml` in the current directory.[11]

Give this file the following content:

```
version: '3.7'

services:
  hello-world:
    image: python:3.8-alpine
    ports:
      - "80:8000"
    command: "python -m http.server 8000"
```

The file defines a container with the name `hello-world`, based on an image `python:3.8-alpine`, where port 80 from the Raspberry Pi is forwarded to port 8000 from the container.

Finally, the command `python -m http.server 8000` is run in the container, which serves the container's root file system on a web server running on port 8000.

Save the file with Ctrl+o and exit nano with Ctrl+x. It's a good practice to always check the syntax of your Docker Compose file. You can do this with:

```
docker-compose config -q
```

11 While nano is an easy editor to use for beginners, if you're serious about using Raspberry Pi OS and Linux in general I recommend you to use a more powerful command line text editor, such as Vim or Emacs. Both editors are highly configurable and can be extended with plugins, for instance for automatic syntax highlighting. I use Vim for everything that I write: code, configuration files, as well as texts. In fact, this complete book has been written in Vim.

If you get no output, the Docker Compose file in the current directory passes some basic checks. But I recommend doing some extra checks with the `yamllint` command, which can check all sorts of YAML files, not only Docker Compose files. Install it like this:

```
sudo apt install yamllint
```

Then check your Docker Compose file with:

```
yamllint docker-compose.yml
```

Warnings are shown in yellow: those are tolerable. Errors are shown in red: those should be corrected. The numbers at the beginning of the line are the line and column numbers that concern the error or warning. Look at your file at that location, fix the error, and then check the syntax again.

If your Docker Compose file has valid syntax, start up the container:[12]

```
docker-compose up -d
```

Now, open your web browser and visit the IP address of your Raspberry Pi. You should see a directory listing such as this one:

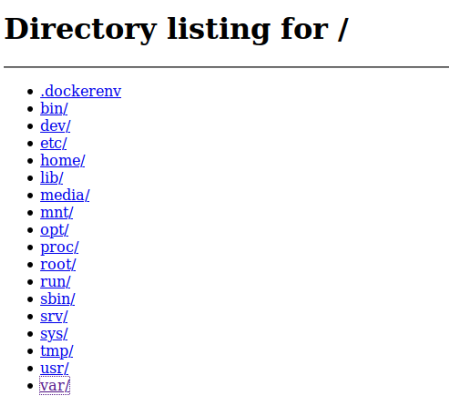

Figure 2.14 Docker Compose has started the Python container, which runs a web server.

12 If you want to use another name for your Docker Compose file, for instance if you want to maintain two versions of your container setup, use the `-f FILENAME` option to the `docker-compose` command. For instance: `docker-compose -f docker-compose-alternate.yml up -d`.

If this works, take the container down:

```
docker-compose down
```

You can verify that there's no container running anymore:

```
docker ps
```

Normally this should list all running containers. If no container is running, you should see an empty list.

Finally remove the YAML file, because it was just an example to illustrate how Docker Compose works:

```
rm docker-compose.yml
```

If working with Docker Compose seems convoluted for now, give it some time to grow on you. The advantage will become much clearer after a few chapters. At the end of this book, you have your complete home automation setup defined in one `docker-compose.yml` file and one directory. This means you can easily create a backup without the risk of forgetting files. And you can easily recreate your whole home automation setup in a few minutes on another Raspberry Pi if you want to move it to other hardware.

2.10 ● Summary and further exploration

In this chapter you learned which Raspberry Pi models are suitable for home automation and what the requirements for a reliable home automation gateway are. This is the foundation upon which the rest of this book builds.

Then you installed Raspberry Pi OS Lite on your Raspberry Pi, learned to set up network connectivity and SSH for remote access, and the basic setup steps to get your Raspberry Pi up and running. There's much more to say about Raspberry Pi OS. If you want to feel more comfortable on the command line of Raspberry Pi OS, I recommend you to buy an introductory Linux book.

I also talked about the tmux terminal multiplexer. While it's not strictly necessary, you'll soon find yourself executing many long-running commands and doing various tasks at the same time. Tmux helps you to do this more efficiently: you can run multiple terminal sessions in one window and even copy and paste text with a few keyboard shortcuts.

Because I use the Python programming language in this book, I also introduced you to its package manager pip and Python virtual environments. Another important program I'm using in this book is Docker: it's an easy way to run various services by running each service in a separate container, isolated from all other containers. I'm especially a big fan of Docker Compose because it lets you specify all your containers in one file.

There's a lot more to tell about Docker and Docker Compose, but the basics are in this chapter, and in the various chapters in this book, I'll add more when you need to know it.

Ultimately, the main takeaway of this chapter is this: the Raspberry Pi is a powerful platform to create a home automation gateway, and there are almost no limits to its flexibility.

Chapter 3 ● Secure your home automation system

Security is an important topic in home automation. You don't want someone else controlling your house or spying at home. That's why I advocate a fully open-source and self-sufficient solution in this book. But these are just architectural choices that tend to result in a more secure system. The devil is in the details, and the security of your system depends on a lot of parameters.

People tend to forget that a home automation gateway, especially a Raspberry Pi, is a full-blown computer. The advantage is that a Raspberry Pi can do a lot, but the disadvantage is that it can be misused to do a lot of things in your house that you don't want.

In this chapter, I'll explain some steps you can take to keep your home automation system secure: separating home automation devices on their network, user management, encryption and authentication, keeping your packages up-to-date, and so on.[1] I don't cover all these topics in detail, because then this book would become a network and Linux security book. But I'll explain enough to make your setup 'reasonably secure', which means more secure than 99% of the home automation systems in the world.

Warning:
There's no such thing as 100% security. Software can be broken, people can make mistakes, or you can just have bad luck. Even I had one of my servers been broken into recently because I was lax with updating a web application on my webserver. And that's after writing about Linux security for 20 years and after passing the LPIC-3 303 Linux Enterprise Professional - Security exam. So you should expect that one day you'll face malware or a break-in. The only thing you can do is make sure that the attacker will face continuous hurdles and won't be able to do much when he's in your system.

3.1 ● Some general computer security principles

There are a lot of general computer security principles, and writing about them would take a complete book, but the topic is important enough to at least give you a general overview of the things you should think about concerning the security of your home automation system. I can't address all these principles in detail, but this way you know which concepts you should read about when you're interested in more information. I'll refer to some of these principles in the other sections of this chapter, where I talk about some concrete steps you can take to secure your home automation system.

1 Note that I talk about your home automation system here. That is, while this chapter (and this book) focuses on your home automation gateway, you also have to think about the security of your other devices and your network.

Note:
Most of these security principles are really common sense, so it's tempting to brush them aside. Many of the biggest security vulnerabilities are the result of developers violating these 'common sense' principles, so they are not as common as you would think. As ordinary as these principles sound, please take a moment to think about them!

Minimize the attack surface
When security experts talk about attack surface, they mean everything that can be attacked. In theory, this is everything. So what this principle comes down to in practice is: install only the software you need and nothing else, and disable functionality that you don't need. You already encountered this principle in Chapter 1 when I urged you to install the Lite version of Raspberry Pi OS on your Raspberry Pi.

Keep It Simple Stupid (KISS)
This principle is also known as: "Complexity is the worst enemy of security". The more complex your setup, the easier you'll make mistakes, forget to update a system, or you just don't understand the security ramifications of your setup.[2] The same principle holds for specific software: using more complex software is a bigger risk (and has a bigger attack surface) than using simple software. Keep your setup and your programs as simple as possible.

Security is as strong as its weakest link
If you're installing highly secure software, keeping your packages up-to-date, running a firewall, encrypting all network traffic, and so on, you're maybe confident enough to make your Raspberry Pi available on the internet so you can control your devices at home when you're outdoors. But this won't hold off attackers if you're using the Raspberry Pi's default password or another very simple password. The whole chain of your security breaks as soon as someone breaks the weakest link, in this case, a weak password.

The principle of least privilege
Each component and user should have the least privileges possible. If you want to follow this principle to the letter, each program should be able to do exactly what it needs and nothing more, and the same should hold for users. The reason is simple: if one system or one user account has been compromised, it will be used as a stepping stone to attack other systems. The fewer privileges the compromised system or user has, the less an attacker can abuse the system to attack the rest of your network.[3]

Defense in depth
This is also called the castle approach. The idea is that in a Medieval castle the lord has

2 This is actually a risk with the DIY approach I'm advocating in this book. If you don't keep this principle in mind while enthusiastically building your home automation system, you risk ending up with a complex amalgam of various interconnecting systems that you're not equipped to handle.

3 This is a principle that is very difficult to follow to the letter for specific programs, also in this book. If you want to know more about ways to implement this, read about mandatory access control (MAC), implemented by systems such as SELinux and AppArmor.

various layers of security to protect him from attackers: water, outer walls, inner walls, and so on. In your home automation system, you should do the same. Don't trust only one security measure such as a password to hold off attackers, but also implement IP blacklisting, a firewall, TLS, and so on. If one layer of security is broken, the other ones are still there to protect you.

3.2 ● Isolate your home automation devices

Insecure home automation devices can be used as a stepping stone to attack the rest of your network. One of the most powerful ways to decrease this risk is to isolate these devices on their network. This is following the principle of least privilege: your home automation devices shouldn't have access to your whole network. So if you isolate your home automation devices and someone manages to break into one of them, he doesn't have access to your main network, which contains your computers, smartphones, tablets, and so on.

There are a couple of ways to isolate your devices, but they all need some network infrastructure, which is out of scope for this book, so here's a general overview of how you could do this.

3.2.1 ● Physical isolation

The most drastic way to isolate your home automation devices is to put them on a network that is physically isolated from the rest of your network, so there's no link (or a gatekeeper) between them. In many cases, complete physical isolation is not possible or desirable, because this means that you don't have access to your home automation devices or your home automation dashboard from your computer and smartphone.

However, if you're particularly security-conscious, it is possible. You connect your Raspberry Pi home automation gateway and all your networked home automation devices to one switch, router, or wireless access point, which you connect to one Ethernet port of your main router/firewall. Your main home network is connected to another Ethernet port of your main router/firewall. And with the right firewall rules in your router, you forbid all network traffic between both networks. This way your home automation gateway still has internet access, which is needed for software updates and to collect information from internet sources, but it doesn't have access to the rest of your network.

Of course, interacting with your home automation gateway becomes much more troublesome this way, because the rest of your network doesn't have access to your home automation network either. You can't connect to your Raspberry Pi from your computer using SSH, because you're not on the same network. You could (temporarily) connect a keyboard and monitor to the Raspberry Pi each time you have to administer it, but that's not very user-friendly.

So if you choose full physical isolation, it's a good idea to connect a dedicated tablet to your home automation network that you only use to administer your home automation gateway

and to view your home automation software's dashboard. For instance, you can mount this tablet on the wall in your living room and let it run the browser app in fullscreen mode to show the dashboard, and when you need to administer your gateway you take it from the wall, connect a USB keyboard and use an SSH app to connect to your gateway and enter commands.[4]

Many networked home automation devices will probably use Wi-Fi instead of Ethernet these days. In this case, you only need a dedicated access point for your home automation devices, preferably with one or more LAN (Ethernet) ports, so you can connect your Raspberry Pi gateway using an Ethernet cable, and your other devices such as Shelly or Sonoff devices or your dedicated home automation tablet connect to your access point using Wi-Fi. The WAN port of your access point then goes to one of the LAN ports of your main router/firewall. Make sure to firewall both networks in your main router, and check that they can't access each other. A dedicated access point is a cheap and easy way to create an isolated network for your home automation devices.

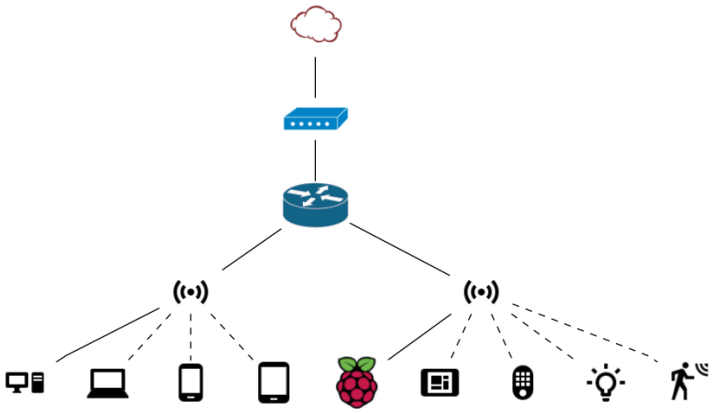

Figure 3.1 An isolated network for your home automation devices is the best security measure you can take.

Some manufacturers are now selling so-called "IoT routers", specifically developed for this purpose of isolating and securing a home automation network. Most of these are not open-source, and you depend on the manufacturer for updates. If you're fine with that, buying one of these is a good solution, but they're not cheap. In principle, you could also use any general-purpose Wi-Fi access point for this purpose, as long as you can make sure that you can keep it secure and that you update it regularly.

If you have an older Wi-Fi access point that you want to use for your home automation network and it doesn't get any updates anymore from its manufacturer, chances are that you can still use it securely after installing the alternative open-source firmware OpenWrt (https://openwrt.org). This is the system I advocate: it's open-source and it supports a

4 A popular open-source SSH app for Android is ConnectBot (https://connectbot.org), and for iOS there's Blink Shell (https://blink.sh). Termius (https://termius.com) is also a popular choice for both iOS and Android. It's free but not open-source.

lot of Wi-Fi access points, new ones as well as older ones. OpenWrt also has a firewall integrated, so you can further restrict what the connected devices are allowed to do.

If you want to take it up a notch, you can install extra software on OpenWrt to visualize the network traffic on your home automation network. An interesting piece of software is SPIN, which stands for Security and Privacy for In-home Networks (https://spin.sidnlabs.nl). This open-source software for OpenWrt lets you visually inspect live network traffic of devices on your network and you can even block suspicious or unwanted traffic or complete devices in a couple of clicks.[5]

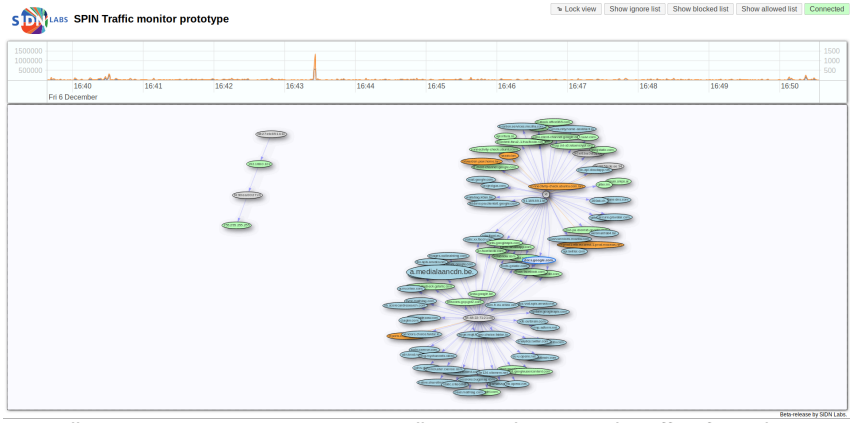

Figure 3.2 Install SPIN on an OpenWrt router to visually inspect live network traffic of your home automation devices.

3.2.2 ● VLANs

A VLAN or virtual local area network is a way to separate network devices that are connected to the same physical network into different 'virtual' networks. This is called 'logical isolation', in contrast to the physical isolation of connecting your home automation devices to a physically separate network than your other devices.

So by using VLANs, you don't need extra cables or access points, but you need a switch that understands VLANs: a managed switch. Entry-level switches don't support VLANs. In many situations, your router also needs to understand VLANs. A router running OpenWrt or OPNsense (https://opnsense.org) supports VLANs.

There are various options to get your Wi-Fi devices on a specific VLAN. You could get a separate Wi-Fi access point for your home automation devices and assign it to your home automation VLAN by connecting it to a port on your switch with that VLAN ID. This way all the network traffic from that access point is in that VLAN. Or you could set up separate

5 The easiest way to install SPIN is by installing the Valibox firmware (https://valibox.sidnlabs.nl) on supported hardware, such as the GL-Inet AR-150 or GL-Inet MT300A travel routers, or even on a Raspberry Pi 3.

SSIDs on a Wi-Fi access point: one for your home automation devices and another one for your main network devices. You can then assign each SSID to a separate VLAN. However, not every access point supports multiple SSIDs.

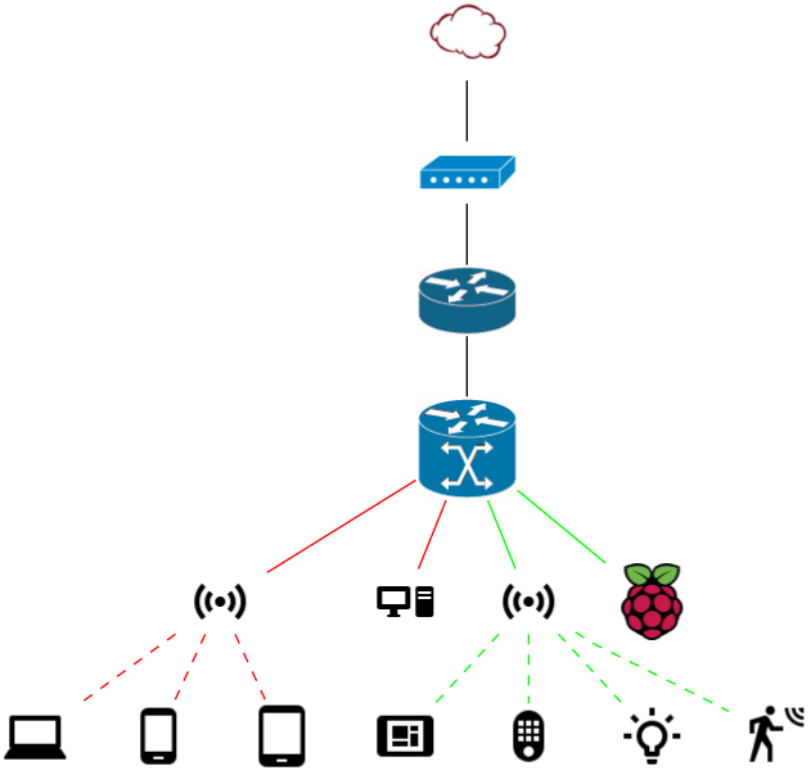

Figure 3.3 With a managed switch you can logically isolate different parts of your physical network: the red connections form one VLAN and the green connections another one.

3.2.3 ● Firewalls

Another way to isolate your home automation devices is by setting some firewall rules. A firewall is software that blocks or allows network traffic based on specific rules. For instance, you could configure your Wi-Fi lamp to talk to your home automation gateway and your router, but not have access to the internet, nor any other devices on your network.

A firewall can be run on any network device, but often it runs on your main router that connects all your local network devices to the internet. If you're running a combined router/modem of your internet provider, that's the central place where you could configure firewall rules. If you're running your router (for instance using OpenWrt or OPNsense) that is connected to your internet provider's modem to connect to the internet, your firewall rules should be configured on your router device.

You can also combine a firewall on one or more devices with the other types of network isolation. For instance, you could isolate all your home automation devices using a separate Wi-Fi access point, and then run a firewall on this access point that further restricts the network connectivity of all devices on your home automation network. Maybe you want to restrict your dashboard tablet's traffic so that you are only able to use it to connect to your dashboard software running on your home automation gateway, as well as some other traffic that you need for updates, DHCP, NTP and so on.

You can even run a firewall on a device such as your home automation gateway. On the Raspberry Pi, you can do this with the `iptables` command, or the more user-friendly `ufw` command. This way you can further restrict what devices can connect to your Raspberry Pi. Even if you have a firewall running on your router, another firewall on your home automation gateway is a sensible move, according to the defense in depth principle.

As `iptables` has a complex syntax, let's use `ufw` instead on your Raspberry Pi. The name `ufw` stands for 'uncomplicated firewall'. The program comes from Ubuntu - the Linux distribution that prides itself on being user-friendly. Install it with:

```
sudo apt install ufw
```

By default, the firewall is turned off. You can check this with:

```
sudo ufw status
```

This should show **Status: inactive**. Now enable it:

```
sudo ufw enable
```

You should get the message **Firewall is active and enabled on system startup**. A `sudo ufw status` will show **Status: active now**.

Note:
If you get an error message, for instance, **ERROR: problem running ufw-init**, you have probably updated the Linux kernel without restarting your Raspberry Pi, and ufw tries to load a kernel module that doesn't match the currently running kernel version. Simply reboot and try enabling ufw again.

You can get a more verbose status with:

```
sudo ufw status verbose
```

This should show you additionally that logging is on, and that the default firewall rules are: **deny (incoming), allow (outgoing), deny (routed)**. This means:

incoming
This is network traffic that is coming into your Raspberry Pi from other computers. It's blocked by default (**deny**), which is a sensible move: if you accidentally run an insecure service on your Raspberry Pi, it's not reachable from other computers, so it can't be exploited.

outgoing
This is network traffic that is going from your Raspberry Pi to other computers or the internet. It's allowed by default (**allow**), which makes sense: it would be a herculean effort to deny all outgoing traffic by default and specify case by case which servers your Raspberry Pi is allowed to reach.

routed
This is network traffic that is routed. This is blocked by default (**deny**) because it's only useful on a router and you're using your Raspberry Pi as a home automation gateway.

Note:
The default firewall rules are more complex than what `sudo ufw status` verbose shows. For instance, ufw adds firewall rules to allow ICMP packets (ping), mDNS (Bonjour) and other low-level network traffic. You can see them all with `sudo ufw show raw`.

Now that you have enabled the firewall, it blocks most traffic, including SSH. So if you close your SSH session now, you can't log in again. The first thing you should do is allow SSH. Luckily you don't have to know which ports you have to open to allow SSH, because ufw knows the concept of 'apps'. You can list all available apps with:

```
sudo ufw app list
```

You should see **OpenSSH** in the list. You can add firewall rules for OpenSSH with:

```
sudo ufw allow "OpenSSH"
```

Note:
The quotes around **OpenSSH** aren't required here, but you need them when an app has a space in its name, such as **WWW Secure**.

This gives a message with two lines: Rule added and Rule added (v6), telling you that ufw has added firewall rules for IPv4 and IPv6. If you now ask for a status again, you should see two firewall rules added:

```
pi@pi-red:~ $ sudo ufw allow "OpenSSH"
Rule added
Rule added (v6)
pi@pi-red:~ $ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp (OpenSSH)           ALLOW IN    Anywhere
22/tcp (OpenSSH (v6))      ALLOW IN    Anywhere (v6)

pi@pi-red:~ $
```

Figure 3.4 The status of the firewall shows that you have added two firewall rules for OpenSSH.

This shows you that incoming traffic to port 22 (where the OpenSSH server is running) is allowed from anywhere.

Note:
If you want to know which ports the OpenSSH app rule allows before you allow the app, you can ask this information using the `sudo ufw app info "OpenSSH"` command. This also gives you a short description of the application.

If you later want to remove a firewall rule, this needs an extra step. First show all numbered firewall rules:

```
sudo ufw status numbered
```

This shows a number before each firewall rule. Then you can delete a rule, for instance, rule 1:

```
sudo ufw delete 1
```

You don't need the app rules. You can also add a rule for specific ports, such as:

```
sudo ufw allow 22
```

Or you can allow SSH access only from a specific IP address:

```
sudo ufw allow from 192.168.0.100 port 22
```

If you enable the firewall on Raspberry Pi OS, don't forget to allow traffic to the ports for applications you install.

Warning:
If you publish ports for Docker containers, Docker bypasses the ufw rules so the published ports can be accessed from outside. You can let ufw and Docker play nice with each other, but that requires some work. See https://github.com/chaifeng/ufw-docker for an explanation.

3.3 • User management

Many programs know the concept of a user. You can create multiple users, each of them having specific permissions, having their own data or dashboard, … You must manage these users securely. The most important security aspects of users are permissions, passwords, and lifecycle.

3.3.1 • Permissions

Raspberry Pi OS has a default user, pi, which has quite broad permissions: you have access to the GPIO pins, audio, video, you can run any command with root permissions with `sudo`, and so on. It's important that you give this user as many permissions as it needs, but not any more (the principle of least privilege). In practice, these broad permissions of the `pi` user are needed because you have to be able to manage your system.

By default, the `pi` user on Raspberry Pi OS can run commands with root privileges using sudo without having to enter the password. So if you're logged in on Raspberry Pi OS, you leave your keyboard and someone passes by, he has full access to your system. To prevent this, you can change the `sudo` configuration:

```
sudo visudo /etc/sudoers.d/010_pi-nopasswd
```

This file has this line by default:

```
pi ALL=(ALL) NOPASSWD: ALL
```

Change this to:

```
pi ALL=(ALL) PASSWD: ALL
```

Note the difference: PASSWD instead of NOPASSWD. After you have saved this configuration, every time you use sudo you will be asked your password.

Other programs can have their own permissions. Many programs know two types of users: an administrator and a normal user. An administrator can manage the system, add, remove, and edit users, and has full power over the system. Normal users have limited permissions: they can use the system, but not manage it.

In this book, I use some software where user accounts and permissions are possible, but I don't cover these permissions in detail. If you want to set specific permissions, you have to look it up in the program's documentation.

3.3.2 ● Passwords

One of the things you have to give some thought to is your password. The default password of the `pi` user in Raspberry Pi OS is `raspberry`, and I already urged you to change it in Chapter 2. If you keep this default password or change it to an easily guessable one, a compromised device on your network can log into your Raspberry Pi. You want to avoid that.

There are a lot of ways to choose a strong password, but keep in mind that you still have to be able to remember it. The classical rules for a good password are:

- Use at least 12 characters.
- Use a mix of upper and lower case letters, numbers, punctuation, and special characters.
- Don't use an existing word, name, or date.

However, if you try to come up with a password according to these rules, you get something like N%2SKYsjs%#W2PBV7%Aq. Are you able to remember this?

A better way is Diceware (http://world.std.com/~reinhold/diceware.html), which is a

method for picking passphrases using dice to select words at random from a word list. How does this work? Each word in this list is preceded by a five-digit number, with all these digits between one and six. This means that five dice rolls are enough to select a word from this list. A passphrase with six of these words is quite secure, and for better security, you select seven or eight words.

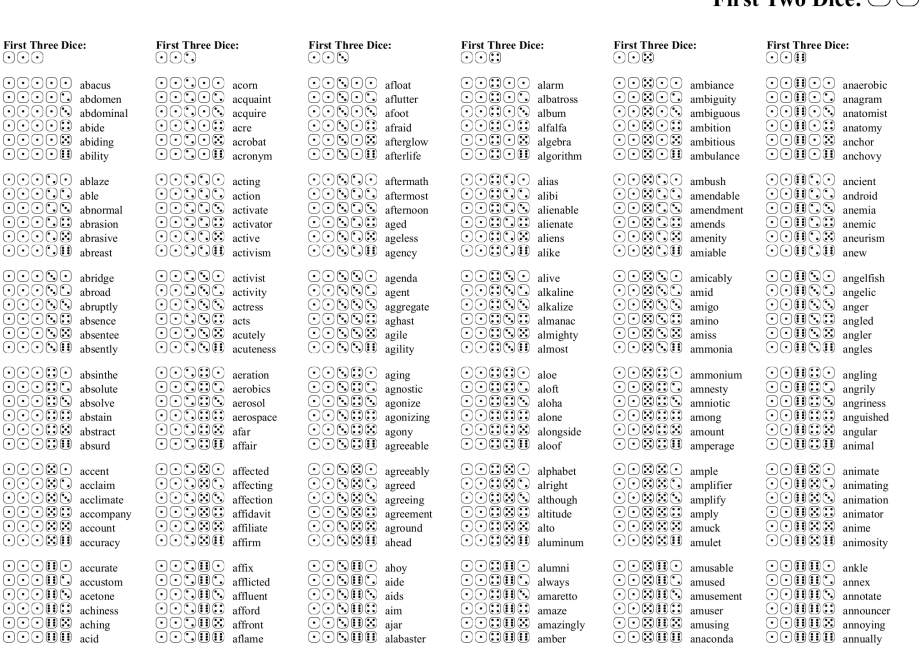

Figure 3.5 Diceware is a strange but very effective way to generate secure passwords that are quite easy to remember. This is a printout by the https://github.com/diafygi/diceware-prettyprint project.

The upside of this rather arcane password generating process is that Diceware passwords are quite easy to remember, such as "lather creasing boastful gradation opal verbally". And although it contradicts the classical rule of not using existing words, it's quite hard to crack because it's so long. A brute-force attack attempt on this six-word Diceware password would take thousands of years. If you're paranoid, just add another word to get an attack time of tens of millions of years, and another one for hundreds of billions of years.

The same rules hold for all your accounts. For instance, if one of the home automation programs that you install, such as the Mosquitto MQTT broker, Node-RED or Home Assistant, offers accounts, you should use them and secure them with a strong password.

Generally, it's not a good idea to use the same password for various services, because if your password for one service is compromised the attacker also gets access to your other services. But of course, it's difficult to come up with a unique strong password for many services. In this case, you should use a password manager, such as Bitwarden (https://

bitwarden.com). You get access to this password manager with one 'master password', and the program stores all your other passwords. If you find this too much of a hassle for all these accounts on your home automation gateway, a compromise could be that you use the same password on all these accounts for Raspberry Pi OS, Mosquitto, Node-RED and so on, but don't share it with other services such as your email account.

3.3.3 ● Lifecycle

Each user account has a lifecycle: you create it, you use it, and you destroy it. However, in practice, the destruction of a user account is regularly forgotten. This could become a weak spot in your security: if you have forgotten that there's a user account lingering on your system and if it has broad permission, someone breaking into it can get access to your system.

That's why you should make it a habit to regularly check whether all accounts you have created are still used. If not, then destroy the account. Each user is a possible entry point into your system for an attacker.

One situation you should check is when you create a test account to test a new system. After your test has succeeded, you create a regular account that you use. It's easy to forget to destroy the test account then, and the risk is compounded when you chose an easy password such as 'test' for this test account because you thought you would destroy this account anyway.

3.4 ● Encryption

If your Wi-Fi access point or one of your other network devices has been compromised, it could snoop on your network traffic. That's why it's important to use encryption as much as possible for your network connections, even on your local network: someone snooping on your network traffic doesn't see the content of your traffic then, but only some garbled data.

You already encountered an encrypted communication protocol in the previous chapter: SSH. You use it throughout this book to log into your Raspberry Pi securely. SSH is so easy to use that it's a no-brainer: you don't want to use another way to log in to your Raspberry Pi remotely.

3.4.1 ● Your threat model

There are two situations now, which are called threat models in security language:

You assume that your local network is trusted
If you trust your local network, it's not that important to have all traffic between devices on your network encrypted. This can save you some work because setting up encrypted traffic is not always that simple. You should still encrypt your traffic with the outside world, as you can't trust it, but that's out of scope for this book.

You assume that your local network is untrusted
If you don't trust your local network, you should treat all network traffic on your local network as internet traffic, and thus encrypt it. This is more work to set up, but it's an extra hurdle for attackers if they manage to break into your network.

In this book, I assume more or less the second situation. I set up encryption (and authentication) for all services that I describe.[6] If you fully trust your local network, you can skip these steps.

3.4.2 ● TLS

Probably the most widely deployed encrypted communication protocol is TLS (Transport Layer Security), previously (and even now still) known as SSL (Secure Sockets Layer). This is generally used as part of HTTPS (HyperText Transfer Protocol Secure) to secure communication to websites, but it can also be used as an encrypted wrapper around other protocols, such as MQTT. The secure version of a protocol commonly gets an S at the end of its name. So MQTTS is MQTT encrypted with TLS, just like HTTPS is HTTP encrypted with TLS.

TLS is not only about encryption, but also about authentication: the server you connect to shows a certificate, which 'proves' that the server you connect to is the server that you think you connect to. This proof is given by a CA (Certificate Authority), that digitally signs the server's certificate. Your operating system has a store of CA certificates it trusts, and when it connects to a server with a TLS certificate signed by one of these CAs, it trusts this connection; otherwise, it gives a warning or an error.

The problem with TLS on your local network is that you have to sign the certificates of your servers yourself. So you have to create a TLS certificate for each of your servers and then do one of the following:

Trust all the TLS certificates
You have to manually trust all your TLS certificates on all your client devices.

Trust your own CA
You have to create your own CA, let this CA sign all your TLS certificates, and manually trust your own CA on all your client devices.

The second way is more maintainable because you only have to trust one CA once on each of your client devices. After this, you can add new services on various devices with new TLS certificates and they are automatically trusted by your client devices when they're signed by your CA.

6 Some of the services I describe don't support encryption and/or authentication directly. In the appendix at the end of this book I describe a way to secure them.

Note:
TLS is used to create a certificate for a domain. This could be the domain of a public website, but also a local domain for one of your computers on your local network. If you want to use TLS on your local network, make sure you use hostnames and preferably set up a local domain in your router's settings. In this book I use the domain **home** for all computers on my network, so they have names such as **raspberrypi.home**, **nas.home**, **laptop.home**, and so on.

3.4.3 ● Setting up your own CA with mkcert

Setting up your own CA can be as simple or complex as you want. For the use case of this book, a home setup with only a handful of devices that need a TLS certificate, I find mkcert (https://github.com/FiloSottile/mkcert) a good choice. This program lets you create a CA certificate and create and sign TLS certificates for your devices. It doesn't need any configuration, and it runs on Windows as well as macOS and Linux. In this section, I assume that you run mkcert on your Raspberry Pi that is your home automation gateway. If you make another choice, you just have to copy some of the certificate files to the right device.

Unfortunately, mkcert is not in the Raspberry Pi OS package repository yet, so you have to install the package manually. Visit its releases page (https://github.com/FiloSottile/mkcert/releases) and search for the ARM version of the latest release. At the moment of writing this book, this was mkcert-v1.4.1-linux-arm. Right-click on the version and copy the link.

Then on your Raspberry Pi, download the version with `wget` and make the program executable. For the above version this is:

```
wget -O mkcert https://github.com/FiloSottile/mkcert/releases/download/
v1.4.1/mkcert-v1.4.1-linux-arm
chmod +x mkcert
```

To check whether it's installed correctly, run it with the `--help` option to see the usage description:

```
./mkcert --help
```

This should show you the available commands and options.

```
pi@pi-red:~ $ ./mkcert --help
Usage of mkcert:

        $ mkcert -install
        Install the local CA in the system trust store.

        $ mkcert example.org
        Generate "example.org.pem" and "example.org-key.pem".

        $ mkcert example.com myapp.dev localhost 127.0.0.1 ::1
        Generate "example.com+4.pem" and "example.com+4-key.pem".

        $ mkcert "*.example.it"
        Generate "_wildcard.example.it.pem" and "_wildcard.example.it-key.pem".

        $ mkcert -uninstall
        Uninstall the local CA (but do not delete it).
```

Figure 3.6 Mkcert is a simple tool to create TLS certificates for your local network.

3.4.4 ● Creating a CA

First, create your CA:

```
./mkcert -install
```

Note:
Don't think there's something wrong when the program seems to be stuck: even on a Raspberry Pi 4, it takes more than 10 seconds to generate the keys for the CA.

This will generate the CA keys, save them in `/home/pi/.local/share/mkcert/rootCA-key.pem` (for the private key) and `/home/pi/.local/share/mkcert/rootCA.pem` (for the public key), and ask for your password to install the latter in the Raspberry Pi OS trust store (`/usr/local/share/ca-certificates/`).

```
pi@pi-red:~ $ ./mkcert -install
Created a new local CA at "/home/pi/.local/share/mkcert" 💥
Sudo password:
The local CA is now installed in the system trust store! ⚡

pi@pi-red:~ $ ls -la .local/share/mkcert/
total 16
drwxr-xr-x 2 pi pi 4096 Mar 29 14:20 .
drwx------ 4 pi pi 4096 Mar 29 12:00 ..
-r-------- 1 pi pi 2484 Mar 29 14:20 rootCA-key.pem
-rw-r--r-- 1 pi pi 1598 Mar 29 14:20 rootCA.pem
pi@pi-red:~ $ ls -la /usr/local/share/ca-certificates/
total 12
drwxr-xr-x 2 root root 4096 Mar 29 14:24 .
drwxr-xr-x 5 root root 4096 Feb 15 21:22 ..
-rw-r--r-- 1 root root 1598 Mar 29 14:20 mkcert_development_CA_301948876062571909438496821762773715328.crt
pi@pi-red:~ $
```

Figure 3.7 Root CA certificate will be used to sign TLS certificates.

Now the operating system on your Raspberry Pi trusts your CA, but you need this trust too on all your devices that want to use TLS to communicate with your Raspberry Pi: your laptop, your smartphone, and so on. To make this happen, you have to copy your CA

certificate to your other device.

Let's see how this goes on a Windows laptop. Open a command prompt in Windows and copy the file from your Raspberry Pi to your computer with the `scp` command from OpenSSH:

```
scp pi@IP:.local/share/mkcert/rootCA.pem .
```

Instead of `IP`, use the IP address or hostname of your Raspberry Pi. The dot (.) at the end of this command is the current directory. So make sure you know in which directory you run this command, so you find the copied file afterwards.

Now you have to add the CA certificate to your web browser. In Firefox you open the menu **Edit** and then **Preferences** (or the page **about:preferences**) and then go to **Privacy & Security**. Scroll towards the bottom and click on **View Certificates...**.

The window you're seeing now is the **Certificate Manager.** This shows the list of all certificate authorities that Firefox trusts. Click on **Import...** and then select the `rootCA.pem` file that you copied from your Raspberry Pi.

Before you trust the CA, click on **View** to examine the certificate. You're seeing a lot of information about the CA certificate, including an SHA-256 fingerprint:

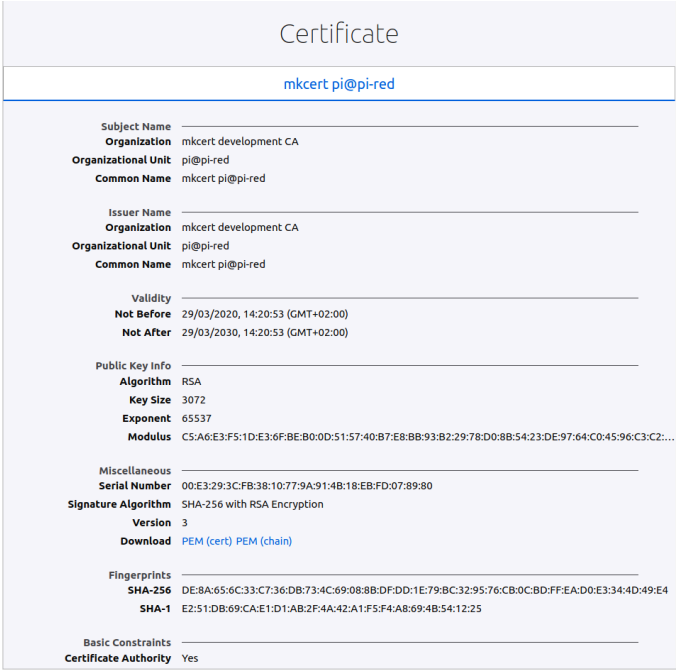

Figure 3.8 Always examine a CA certificate before you trust it.

Now enter the following command on your Raspberry Pi:

```
openssl x509 -in ~/.local/share/mkcert/rootCA.pem -noout -sha256 -fingerprint
```

Compare this fingerprint to the one in Firefox. If they match, enable **Trust this CA to identify websites** in the import window of Firefox and click **OK**. Afterwards, you should see your CA certificate in the list, under **mkcert development CA**. Restart Firefox to make it trust all certificates signed by this CA.

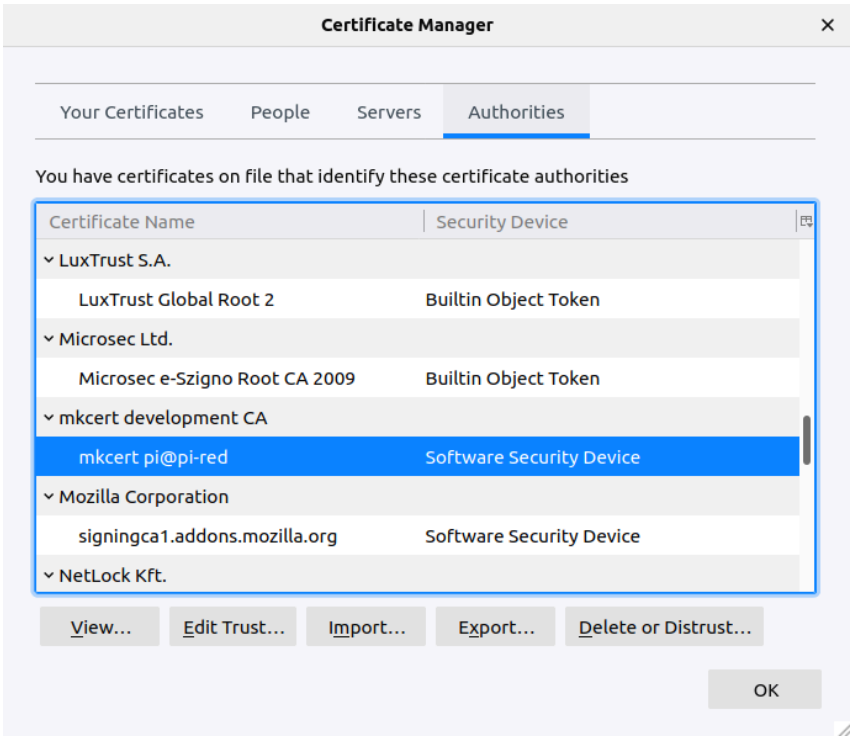

Figure 3.9 After trusting your CA certificate, as added to the list of CAs in Firefox.

3.4.5 ● Creating and signing a certificate

Now that you have created a CA and trust it on your clients, you can create a TLS certificate for the services on your Raspberry Pi. Use your Raspberry Pi's hostname and optionally add the hostname with the domain of your local network. For instance, my Raspberry Pi has the hostname **pi-red** and the domain of my local network is **home**, so I create a certificate with the following command:

```
./mkcert pi-red.home pi-red
```

Generating the keys takes a while, and afterwards the program shows you where it has created the private key (for instance `pi-red.home+1-key.pem`) and the certificate (for instance `pi-red.home+1.pem`).

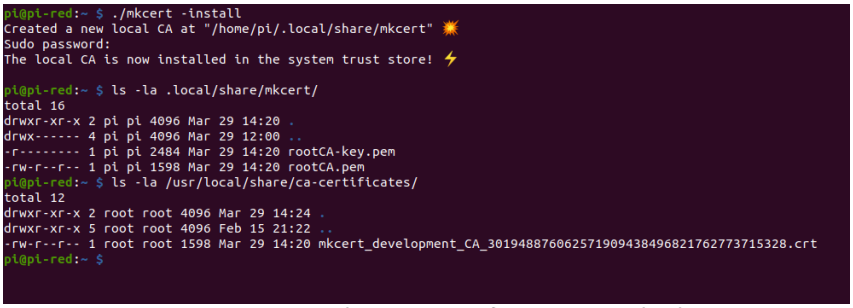

Figure 3.10 Creating and signing a certificate is easy with mkcert.

Now to make it easier to use the TLS certificates in your services, copy the relevant files (the key, the certificate, and the CA certificate) to a separate directory:

```
mkdir -p /home/pi/containers/certificates
cp pi-red.home+1-key.pem ~/containers/certificates/key.pem
cp pi-red.home+1.pem ~/containers/certificates/cert.pem
cp ~/.local/share/mkcert/rootCA.pem ~/containers/certificates
```

How you use these certificates depends on the services you start. I'll explain this in the rest of this book in the relevant sections.

3.4.6 ● Keeping your root CA key secure

Currently, mkcert doesn't secure the root CA certificate you create with a password.[7] This means that everyone who gets access to your Raspberry Pi, can find your root CA key in `~/.local/share/mkcert/rootCA-key.pem` and sign certificates with it that are trusted by your devices. According to mkcert's developer, mkcert is meant to be used for development certificates, not for "production use". However, for this book, I find mkcert a convenient way to add TLS certificates.

I suggest some extra measures to keep your root CA key secure. I have shown here how you generate certificates on your home automation gateway, but it's better to do this on a separate Raspberry Pi or another computer that is not connected to a network. Then you

7 https://github.com/FiloSottile/mkcert/issues/122

put the generated server certificate and key file on a USB stick and copy it to the device it's meant for.

You can reach even better security if you encrypt your root CA key and decrypt it before each use. However, that's out of scope for this book. Consult some online documentation about OpenSSL (https://www.openssl.org) if you want to know more about this.

3.5 ● Keeping your software up-to-date

Software is never ready. And I'm not only talking about features, in this chapter I'm talking about security. Every piece of software contains security vulnerabilities, and each day many of these are found. The developers then release a new version that fixes the vulnerability, but that's no use if you don't update your software to this new version.

That's why it's important to keep all your software up-to-date, not only on your computer and mobile devices but also on your home automation gateway and other devices. The Raspberry Pi that you use as your home automation gateway is the digital centre of your home, so you must keep it up-to-date to fix security vulnerabilities as soon as they are discovered.

3.5.1 ● Update apt packages

The basic system packages and some of the ones that I use in this book are installed with the `apt` package manager. Just entering a `sudo apt instal foobar` to install a package and then forgetting about it is not very secure.

To update all packages that you have installed with `apt` to their latest available version, you first have to update the available information about the newest packages:

```
sudo apt update
```

It's important to think about what this does: this command doesn't update the packages on your system; it updates the information it has about all available packages.

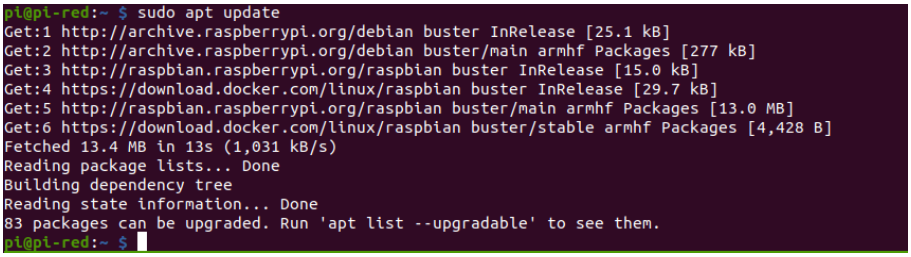

Figure 3.11 Update the information about available package versions with `apt update`.

So now that Raspberry Pi OS has updated its information about the available package versions, the next step is to upgrade the outdated packages. At the end of the `apt update` command, you already get a message that you can list the upgradable packages with the `apt list --upgradable` command. You can type this command to have a look, or just upgrade them with:

```
sudo apt upgrade
```

This will first show you which packages will be upgraded, which new ones will be installed (because they are needed by newer versions of packages you have installed), how much data will be downloaded, and how much additional disk space will be needed. It then asks you for your confirmation. Press **Enter** to start the upgrade process.

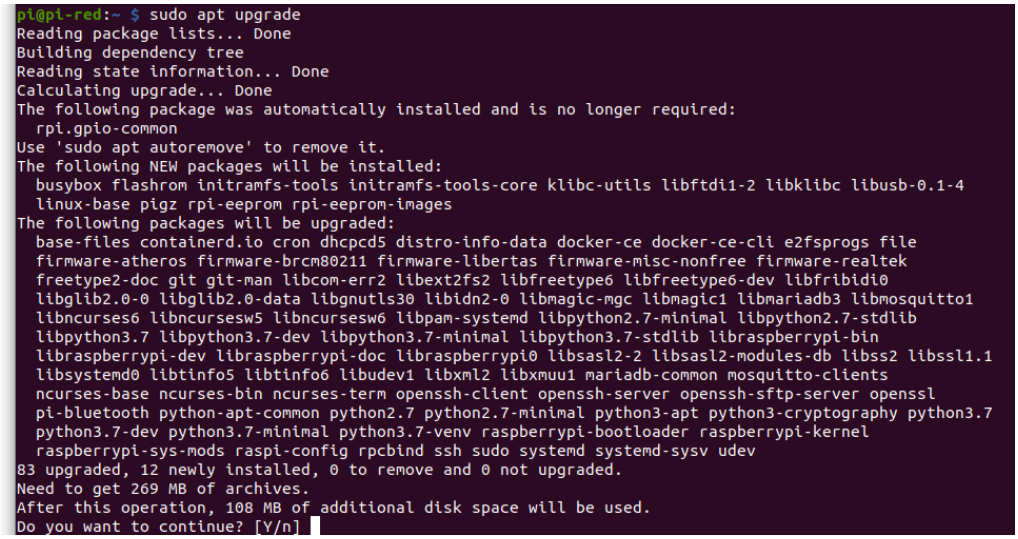

Figure 3.12 Upgrade all packages to the latest available versions with `apt upgrade`.

After all the updates have been downloaded and installed, your operating system is up-to-date again.

Of course, you can now regularly execute a `sudo apt update && sudo apt upgrade` command to update your packages, but the chances are that you will forget this after a while. So another option is to configure Raspberry Pi OS to automatically update all packages.

Warning:
It sounds like a no-brainer to automatically update all packages, but be aware that with each update you risk breaking a working system. So if you depend on your home automation gateway to automatically close your blinds, send you notifications and so on, you risk these things suddenly stopping working because an automatic update has broken something.

Automatically updating all apt packages can be done with the `unattended-upgrades` package. Install it:

```
sudo apt install unattended-upgrades
```

Then open the configuration file:

```
sudo nano /etc/apt/apt.conf.d/50unattended-upgrades
```

Now you see a lot of lines that are commented out with two slashes (`//`). If you ignore these (because they aren't active) and only look at the other ones, you should see something like this:

```
Unattended-Upgrade::Origins-Pattern {
      "origin=Debian,codename=${distro_codename},label=Debian";
      "origin=Debian,codename=${distro_codename},label=Debian-Security";
};
```

However, as you're using Raspberry Pi OS and not Debian, these lines don't apply. You should replace this block by:[8]

```
Unattended-Upgrade::Origins-Pattern {
      "origin=Raspbian,codename=${distro_codename},label=Raspbian";
  "origin=Raspberry Pi Foundation,codename=${distro_codename},label=Raspberry
Pi Foundation";
      "origin=Docker,archive=${distro_codename},label=Docker CE";
};
```

8 How do you know these values? Just look at the output of the `apt policy` command. This gives you the origin, label, codename and archive values you need to fill in. Also read the comments in the beginning of the configuration file of unattended-upgrades.

Now uncomment the following line:

```
//Unattended-Upgrade::Mail "";
```

And fill in the email address where `unattended-upgrades` sends it reports:

```
Unattended-Upgrade::Mail "root";
```

Just use root to send the emails to the local root user. If you have configured Raspberry Pi OS to forward emails for local users to your email address (see Chapter 11), you will receive a report of updated packages in your mailbox.

Save your changes, and after this check `unattended-upgrades` manually with:

```
sudo unattended-upgrades --dry-run --debug
```

If all goes well, Raspberry Pi OS should check for updates daily, install the available updates, and send you an email with a report.

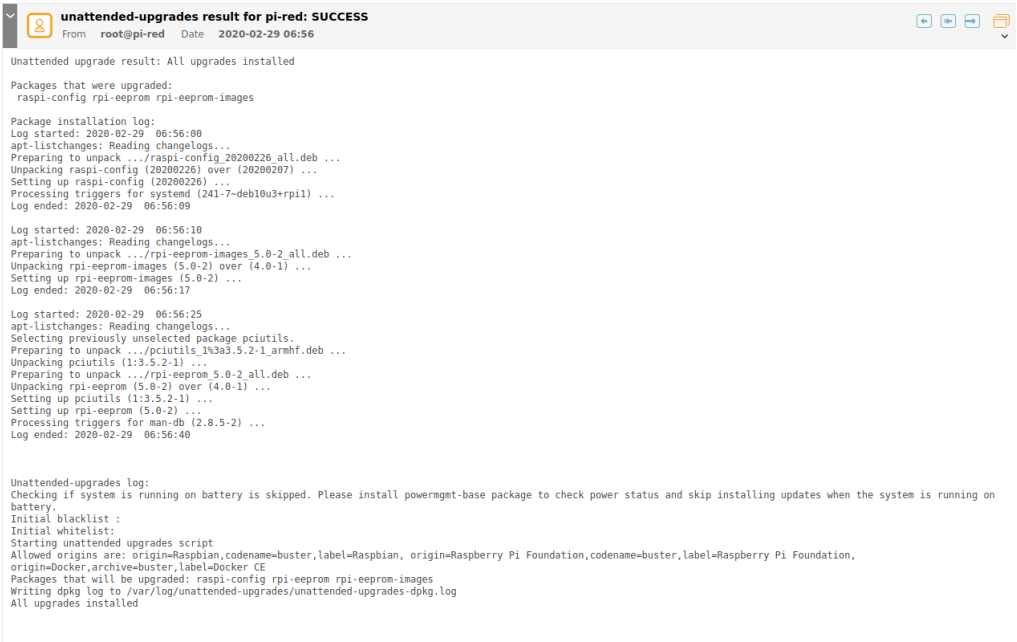

Figure 3.13 The unattended-upgrades package can send you an email on each automatic upgrade it installs

3.5.2 ● Update Docker images

In this book, a lot of services are installed in Docker containers. This means that you have to update these Docker images too. You can see the list of currently downloaded images with:

```
docker images
```

Because I'm using Docker Compose in this book and all Docker containers are defined in the file `docker-compose.yml` in the home directory of the user `pi`, updating the images is as easy as:

```
docker-compose pull
```

The command now downloads (`pull`) the newest version of the images of all containers defined in your Docker Compose file. After this command exits, you can verify this by running `docker images` again. The **Created** column should show today instead of some time in the past.

Now your images have been updated, but if the containers are running they are still using the old version of the image. So you have to restart the containers:

```
docker-compose restart
```

After this command, the containers use the updated images. The older images can now be deleted to free some disk space:

```
docker image prune -f
```

Note:
Some programs can automatically update all your Docker containers, but I won't recommend this because it can break your services if you upgrade to a version with breaking changes in configuration file formats. If you want to do this, you can install watchtower (https://containrrr.github.io/watchtower/) or ouroboros (https://github.com/pyouroboros/ouroboros).

3.5.3 ● Update pip packages

If you've installed Python packages with pip, you can upgrade them too. First upgrade pip itself:

```
sudo pip3 install --upgrade pip
```

Then you can list the outdated packages with:[9]

```
sudo pip3 list --outdated
```

You can upgrade them individually with `sudo pip3 install --upgrade PACKAGENAME`, or upgrade all outdated packages with:

```
sudo pip3 install --upgrade $(sudo pip3 list --outdated|tail -n +3|awk ⏎
'{printf $1" "}')
```

Warning:
Updating Python packages this way may clash with Python packages that you installed as apt packages. This is one of the reasons I prefer to run most software in Docker containers: this will never break the Raspberry Pi OS packages.

If you have created virtual environments, you have to do repeat the same actions in all your virtual environments, but without sudo. That is, activate your virtual environment (see Chapter 2), upgrade pip, upgrade all Python packages, and then deactivate your virtual environment.

3.5.4 ● Update manually installed packages

Sometimes a package is not yet in the Raspberry Pi OS repositories or Python's PyPI packages. You can manually install it then, for instance by downloading the source code and compiling and installing it.[10] But you are responsible then for updating it too. That's why I try to avoid this approach in this book.

9 Drop the `sudo` if you want to upgrade the Python packages that you installed only for the local user.
10 The installation of mkcert earlier in this chapter was an example that required a manual install.

If you do this, make sure to regularly check whether there's a new release and download it. If you're using a program that doesn't have releases, you probably downloaded its source code initially with a `git clone` command and then compiled and installed it. Make sure then to regularly do a `git pull` in the source code directory to download the latest changes from the Git repository and recompile and reinstall it.

3.5.5 ● Update your home automation devices

Your home automation devices also run software, and it's easy to forget this because these devices don't look like a computer. But you must keep them up-to-date too. Generally, the software on these devices is called firmware. So you should regularly consult the manufacturer's website for firmware updates and look up how you install these on your devices.

The same holds for your network devices, actually, such as your routers, Wi-Fi access points, and so on. Consult their manufacturer's websites and update them. Many of these network devices can (manually or automatically) check for updates, so it's quite easy to keep them secure, as long as the device gets updated. If your router or access point doesn't receive updates anymore from its manufacturer, you could try installing OpenWrt on it.

3.6 ● Summary and further exploration

These were quite some pages about security, but I hope you agree with me that it's an important topic to think about when you're building an home automation system. That's also why I have discussed this topic so early in this book.

First, it's important to know that the basics of computer security are not that difficult. They are embodied in general principles such as minimizing the attack surface, Keep It Simple Stupid (KISS), security being as strong as its weakest link, the principle of least privilege, and defense in depth. If you are only able to remember one thing, it should be these five computer security principles.

The next step should be to isolate your home automation devices, be it by connecting them on their own physical network, VLAN and/or with a firewall. User management is also an important topic: have all the user accounts the right permissions and strong passwords and don't you keep them lingering when they're not needed anymore?

Encryption is another important topic. You already used it in the previous chapter to log into your Raspberry Pi from your computer with SSH. In this chapter, I walked you through the foundation of TLS encryption. You'll heavily use this in the rest of this book to encrypt the connection to all your home automation services.

And last but not least I talked about the importance of keeping your software up-to-date. Unfortunately, there's no one-size-fits-all solution to this, and that's why I covered how to keep software updated you installed with apt, Docker, pip, and git.

There's much more to tell about computer security, but with this chapter, you already have a good foundation. If you want to delve deeper into it, I recommend you to buy some books about network security and Linux security. But above all, try to break into your systems: it's the best way to learn how to secure them.

Chapter 4 • MQTT (Message Queuing Telemetry Transport)

In this chapter, you'll learn how to work with MQTT, a lightweight network protocol for data exchange. MQTT is a perfect protocol for a modular home automation system because it's supported by various home automation systems and programs. As such, MQTT works as the 'glue' to connect different parts of a heterogeneous system.

First I'll give you an overview of what MQTT is. Then I show you how to install and configure an MQTT server on your Raspberry Pi, including a secure setup with encrypted communication (using TLS), authentication, and an access control list (ACL). I show you how you send and receive messages with a couple of MQTT clients (both command line and graphical client) and explain how you can write Python programs to communicate using the MQTT protocol.

4.1 • What is MQTT?

The first version of MQTT (which stands for Message Queuing Telemetry Transport) already dates from 1999. The original designers envisioned a protocol that could efficiently use available network bandwidth to send various types of data with all sorts of Quality of Service (QoS). Back in 2009, one of the inventors of MQTT, IBM's Andy Stanford-Clark, already showed the world the potential of MQTT for home automation by linking his home automation system to Twitter using MQTT.

4.1.1 • Central intermediary

The following parties take part in MQTT communication:

- the broker, which acts as a central intermediary that handles all communication;
- clients, who are:
- publishers, who send data using the MQTT protocol
- subscribers, who subscribe to specific topics and receive the relevant messages

You already see that MQTT has its own jargon: traditionally the broker would be called a server, the publishers senders and the subscribers receivers.

A broker can have multiple clients, and a client can be both publisher and subscriber.
The broker works as an intermediary so publishers and subscribers don't have to know of each other's existence. This works like this: each MQTT message has a topic (comparable to a URI for a web page, see the next subsection) and a payload (the content of the message). If a subscriber is interested in receiving the content of a specific topic, it connects to the MQTT broker and subscribes to that topic.

Now when a publisher sends a message to the MQTT broker, the broker forwards the message to all clients that are subscribed to the topic. The clients don't even see which publisher has sent the topic.

Note:
MQTT topics are case sensitive: `foo/bar` is another topic than `Foo/bar`.

4.1.2 ● Hierarchical names

You could consider a topic as a sort of a name that clients can use to publish data or subscribe to receive data. But to bring some order in this naming, MQTT defines topics as hierarchical names: a topic consists of components separated by a slash (`/`), such as in URIs or Linux file paths.

Warning:
There's one essential difference: an MQTT topic never starts with a slash. If you start a topic with a slash, it's a topic with an empty first component.

Apart from being hierarchical, the names can be chosen freely. Applications are free to use their naming conventions. So in contrast to URIs, there's no specific list of top-level domains.

For instance, the home automation system Home Assistant (which I'll cover in Chapter 10) uses the following convention for its MQTT topics about the state of components:

```
<discovery_prefix>/<component>/[<node_id>/]<object_id>/state
```

Here `<discovery_prefix>` defaults to homeassistant (you choose this prefix in your configuration of Home Assistant), `<component>` is the type of component (such as `binary_sensor`, `sensor`, `switch` and so on), `<node_id>` is an optional ID of a node, and `<object_id>` is the ID of the object.

As an example, if you have a temperature sensor in your bedroom that sends its sensor value to MQTT, it could publish a message to the `homeassistant/sensor/bedroom_temperature/state` topic, which would have a value such as `18.7`.

Another architectural proposal to try to bring some conventions in the topic names is mqtt-smarthome (https://github.com/mqtt-smarthome/). It defines some conventions for topics, such as:

toplevelname/status/itemname
: Report states (values from sensors, feedback from actuators).

toplevelname/set/itemname
: Request state changes by publishing a new value.

toplevelname/get/itemname
: Actively request values from a device.

toplevelname/command
: Request to execute a specific command.

toplevelname/connected
: Report the connection status of a program or device.

Various projects follow the mqtt-smarthome conventions. A list can be found on https://github.com/mqtt-smarthome/mqtt-smarthome/blob/master/Software.md.

4.1.3 ● Using wildcards

A very convenient feature of MQTT are wildcards for topics. For instance, a client which is interested in all topics below `homeassistant/sensor/bedroom_temperature`, subscribes to `homeassistant/sensor/bedroom_temperature/#`. Apart from messages with the `homeassistant/sensor/bedroom_temperature/state` topic, the subscriber then also receives messages with the topics `homeassistant/sensor/bedroom_temperature/last_updated`, `homeassistant/sensor/bedroom_temperature/last_changed` and so on.

If a client is interested in all topics of Home Assistant, it subscribes to `homeassistant/#`. And a client that wants to receive all topics of the MQTT broker just subscribes to `#`: this is the ultimate wildcard that covers all topics.

But sometimes you're interested in all topics with a specific component in the lowest level, for instance, all topics below `homeassistant/sensor` that end on `last_changed`. That's where you use the + wildcard. So if you are interested in all messages about the last changes of all sensors in Home Assistant, you subscribe to `homeassistant/sensor/+/last_changed`. Thanks to the hierarchical nature of MQTT topics, this is very easy.

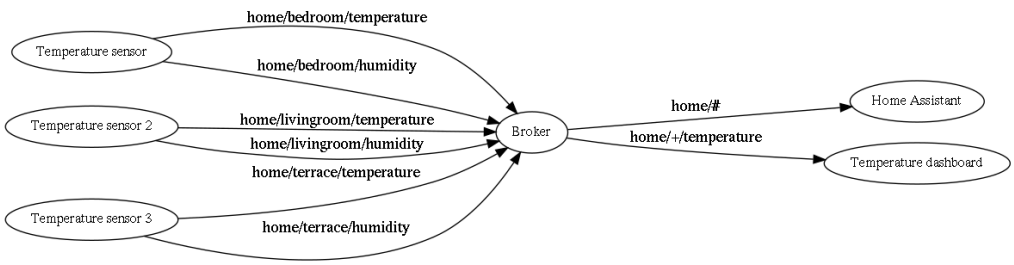

Figure 4.1 Sensors can publish their values to specific MQTT topics, while dashboards and other software can subscribe to topics with wildcards.

4.2 ● Installing and configuring the Mosquitto MQTT broker

A well-known open-source MQTT broker is Eclipse Mosquitto (https://mosquitto.org), which implements MQTT protocol versions 5.0, 3.1.1, and 3.1 and supports plain (unencrypted) MQTT, MQTT over TLS, MQTT over TLS with a client certificate, MQTT over WebSocket and MQTT over WebSocket with TLS.

You're going to install Mosquitto in a Docker container on your Raspberry Pi and use it as the MQTT broker for all other home automation software in this book. But first create some directories for Mosquitto's configuration, data, and logs:

```
mkdir -p /home/pi/containers/mosquitto/{config,data,log}
```

On the next few pages I'll explain the installation and configuration of two setups:

- A basic setup without encryption, authentication or access control for plain MQTT and MQTT over WebSocket
- A secure set up with encryption, authentication, and access control for MQTT over TLS and MQTT over WebSocket with TLS.

Depending on your threat model (see Chapter 3), you choose the first setup if you trust all devices on your network and the second setup if you don't.

Note:
If you created a CA and server certificate for your Raspberry Pi in Chapter 3, the addition of TLS encryption and authentication is not that difficult, so I encourage you to use the secure setup.

4.2.1 ● A basic Mosquitto setup

Even if you want to have a secure MQTT setup, first try a basic setup without encryption and authentication. Create a basic configuration file:

```
nano /home/pi/containers/mosquitto/config/mosquitto.conf
```

Add the following lines in this file:

```
port 1883
listener 9001
protocol websockets
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
```

This configures Mosquitto to both use plain MQTT on port 1883 as MQTT over WebSocket on port 9001.

Now create a `docker-compose.yml` file in your home directory with the following content:

```
version: '3.7'

services:
  mosquitto:
    image: eclipse-mosquitto
    container_name: mosquitto
    restart: always
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./containers/mosquitto/config:/mosquitto/config
      - ./containers/mosquitto/data:/mosquitto/data
      - ./containers/mosquitto/log:/mosquitto/log
      - /etc/localtime:/etc/localtime:ro
    user: "1000:1000"
```

This exposes ports 1883 (for plain MQTT) and 9001 (for MQTT over WebSocket) and forwards the same ports on your Raspberry Pi to the Docker container. It also mounts the directories you created so they become available in the container. At the end you specify that the container is run as the user with user ID 1000 and group ID 1000, which is the `pi` user. This prevents the permissions problem when Mosquitto tries to write to the log file.

Note:
This Docker Compose file also mounts the `/etc/localtime` file on your Raspberry Pi on the same file in the container. This is one way to synchronize the time zone between the Raspberry Pi OS and your container.

Now start the container:

```
docker-compose up -d
```

Your Raspberry Pi is now running an MQTT broker.

4.2.2 ● Testing your setup with the Mosquitto clients

The Mosquitto project also develops two client programs, which are convenient for quick tests. Install them like this:

```
sudo apt install mosquitto-clients
```

This installs two programs:

A publisher: `mosquitto_pub`
This publishes a single message on a specified topic and exits.

A subscriber: `mosquitto_sub`
This subscribes to specified topics and shows all messages that it receives from the broker on these topics.

Now start tmux and open two panes next to each other. In the first pane, subscribe to all topics:

```
mosquitto_sub -t '#'
```

In the second pane, publish a message:

```
mosquitto_pub -t 'test/foobar' -m 'test message'
```

You specify the topic with the `-t` option and the payload with the `-m` option.

As soon as the `mosquitto_pub` command runs (and immediately exits), you see the test message appearing in the first pane. The `mosquitto_sub` command keeps listening for messages until you interrupt it with Ctrl+c.

You entered these commands on the same Raspberry Pi, but you can do the same on other

Raspberry Pis or Linux computers. If you're subscribing or sending to an MQTT broker on another machine, you just have to add the hostname of the broker with the `-h HOSTNAME` option, where you change `HOSTNAME` to the relevant hostname.

Warning:
Because you don't have set up authentication, using Mosquitto with this configuration means that every device on your network can subscribe to all messages sent to the MQTT broker. And because you don't have set up encryption, MQTT traffic could be sniffed by some devices on your network.

The mosquitto_sub command has a couple of other interesting options:

-v
This not only shows the payload but also the topic for each message.

-F
This lets you show you each message in a specific format.

For instance, the following command shows all messages with a timestamp, their topic and payload:

```
mosquitto_sub -F '@Y-@m-@dT@H:@M:@S : %t : %p' -t '#'
```

4.2.3 ● A secure Mosquitto setup

Let's now add encryption, authentication and access control. Change Mosquitto's configuration file `mosquitto.conf` to:

```
# MQTT over TLS
port 8883
cafile /mosquitto/config/certs/rootCA.pem
keyfile /mosquitto/config/certs/key.pem
certfile /mosquitto/config/certs/cert.pem

# MQTT over WebSocket over TLS
listener 9091
protocol websockets
cafile /mosquitto/config/certs/rootCA.pem
keyfile /mosquitto/config/certs/key.pem
certfile /mosquitto/config/certs/cert.pem
```

```
# Authentication and access control
password_file /mosquitto/config/passwords
allow_anonymous false
acl_file /mosquitto/config/acl

# Miscellaneous
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
```

This configures Mosquitto to both use port 8883 for MQTT as well as port 9091 for MQTT over WebSocket.

Note:
Port 8883 is reserved for MQTT over TLS. There's no such reserved port for MQTT over secure WebSocket, so you're free to choose one.

Both for the MQTT as the WebSocket part, you define the location of the CA file, key file, and certificate file for TLS.[1]

After this comes the authentication part. You define the location of the password file, and the line `allow_anonymous false` makes sure that you need to supply a username and password to the MQTT broker to be able to connect. Then you define an ACL file (access control list), in which you limit what topics specific users can subscribe and publish to.

Now create a password file with a user `home`. You can do this with the `mosquitto_passwd` program:

```
docker exec -ti mosquitto /usr/bin/mosquitto_passwd -c /mosquitto/config/⏎
passwords home
```

Note:
I run the mosquitto_passwd program in the Docker container, so the path to the password file is relative to the Docker container too: /mosquitto/config/passwords and not /home/pi/containers/mosquitto/config/passwords.

Enter a password and re-enter it. Now the content of the password file (`/home/pi/containers/mosquitto/config/passwords`) looks something like this:

1 You copied these there in the previous chapter

```
home:$6$OUhCo1IlgIUa0oZt$tAP79TwdJj2o50u86K/
YvyZYtXD7C8bfKP6kggxPCip+Bg30UtdiGVJJLJ4OTjehhOIVIAXBo8l43hH3DOB4QA==
```

This is a hashed version of the password you entered.

In the same way, you can add other users with their password.

Warning:
Don't use the `-c` option to the `mosquitto_passwd` command for the other users: this creates the password file again and overwrites your existing users.

Now create the ACL file:

```
nano /home/pi/containers/mosquitto/config/acl
```

Here you specify what topics each user can subscribe and publish to. This is a way to implement the principle of least privilege (see the previous chapter).

To keep tests in this book easy, I always use the same user, `home`, for all MQTT clients. If you want to give the user home read and write access to all MQTT topics, the ACL file should be:

```
user home
topic #
```

I recommend testing your Mosquitto setup with this ACL file first. If this works, you can refine the ACL. For instance, you could give an app for your voice control system only access to the specific MQTT topics to get the intent behind your voice command and to send a voice message to your text to speech engine (see Chapter 12 for this app):

```
user rhasspy-app-time
topic read hermes/intent/GetTime
topic write hermes/tts/say
```

Of course, you have to create a password for this user then and let the program authenticate to the MQTT broker with the `rhasspy-app-time` username and the relevant password. If this app tries to subscribe to another topic or send a message to another topic than is

specified in the ACL, it won't be able to.

You can also use wildcards in the topic, for instance:

```
user bt-mqtt-gateway
topic write bt-mqtt-gateway/#

user temperature-dashboard
topic read bt-mqtt-gateway/+/+/temperature
topic read rtl433/+/+/+/temperature_C
```

If you don't add `read` or `write` before the topic, Mosquitto will give read and write permissions to the user, such as in the broad `topic #` permissions I gave to the user home earlier.

Note:
I won't create a user and access control list for every application or service in the rest of this book. This all depends on your specific set up and threat model. Just remember to revisit this section from a time when you've configured a service and want to secure it further.

Now change your `docker-compose.yml` file for Mosquitto to:

```
version: '3.7'

services:
  mosquitto:
    image: eclipse-mosquitto
    container_name: mosquitto
    restart: always
    ports:
      - "8883:8883"
      - "9091:9091"
    volumes:
      - ./containers/mosquitto/config:/mosquitto/config
      - ./containers/mosquitto/data:/mosquitto/data
      - ./containers/mosquitto/log:/mosquitto/log
      - ./containers/certificates:/mosquitto/config/certs:ro
      - /etc/localtime:/etc/localtime:ro
    user: "1000:1000"
```

This exposes ports 8883 (for MQTT over TLS) and 9091 (for MQTT over WebSocket over TLS) and additionally mounts the volume with your certificate files.

Note:
The volume with the certificate files is mounted read-only. You don't want them to be changed accidentally.

Now start the container:

```
docker-compose up -d
```

Your Raspberry Pi is now running an MQTT broker with all communication encrypted by TLS, and all clients need to supply the right username and password to be able to connect.

4.2.4 ● Testing your secure setup with the Mosquitto clients

Test this setup again with two panes next to each other in tmux. In the first pane, subscribe to all topics. Try it first as you did with the non-secured setup:

```
mosquitto_sub -t '#'
```

You get the message **Error: Connection refused** because `mosquitto_sub` by default connects to the unencrypted port 1883. So add the right port:

```
mosquitto_sub -p 8883 -t '#'
```

This doesn't show a message, so you could think subscribing worked. But it hasn't, because by default `mosquitto_sub` doesn't use TLS for the connection. To use TLS, you have to add an option with the path of the CA file:

```
mosquitto_sub -p 8883 --cafile ~/containers/certificates/rootCA.pem -t '#'
```

This gives the message **Error: A TLS error occurred**. because you didn't specify the hostname, and the certificate is only valid for the hostname of your Raspberry Pi that you specified while generating it (see Chapter 3). So add it (replace HOSTNAME with the hostname of your Raspberry Pi):

```
mosquitto_sub -h HOSTNAME -p 8883 --cafile ⏎
~/containers/certificates/rootCA.pem -t '#'
```

This gives the message **Connection Refused: not authorised**. That's because you didn't supply a valid username and password. So finally add these:

```
mosquitto_sub -h HOSTNAME -p 8883 --cafile ⏎
~/containers/certificates/rootCA.pem -u home -P PASSWORD -t '#'
```

If you have added the correct username and password, you don't get an error message but the program has been able to subscribe to all topics and is listening to incoming messages. If your username and/or password were wrong, you get the **Connection Refused: not authorised** message again. So now you have verified that only the authorized user can connect to your MQTT broker.

In the second pane, publish a message, with the same parameters for the hostname, port, CA file, username, and password:

```
mosquitto_pub -h HOSTNAME -p 8883 --cafile ⏎
~/containers/certificates/rootCA.pem -u home -P PASSWORD -t 'test/foobar' ⏎
-m 'test message'
```

As soon as the `mosquitto_pub` command runs (and immediately exits), you see test message appearing in the first pane. The `mosquitto_sub` command keeps listening for messages until you interrupt it with Ctrl+c.

Warning:
Make sure you perform these tests with a username that has the permission to subscribe or publish to your test topic. If you use a valid username and password but the user doesn't have the topic you subscribe to in his access control list, `mosquitto_sub` just doesn't show anything, not even an error message. Publishing to a topic your user doesn't have in his access control list has the same effect: `mosquitto_pub` will just exit silently without publishing anything. If you have forgotten that you're using an access control list, this behaviour can be puzzling.

You entered these commands on the same Raspberry Pi, but you can do the same on other Raspberry Pis or Linux computers. The commands are the same, as long as you use the right hostname, and you probably have to change the path to your root CA because you have copied it to another location on the other computer.

Note:
If you want to verify whether the network traffic between the MQTT client on your computer and the MQTT broker on your Raspberry Pi is encrypted, start a network sniffer such as Wireshark (https://www.wireshark.org) on your computer and add the display filter `tcp.port==8883`. You should only see encrypted TLS traffic.

4.2.5 ● Default options for Mosquitto clients

Repeating all those options (hostname, port number, CA file, username, and password) each time you execute the `mosquitto_pub` and `mosquitto_sub` commands is a bit cumbersome. Luckily, you can create a configuration file with default options that will be used if you don't specify them on the command line. First create a configuration file for `mosquitto_sub`:

```
nano ~/.config/mosquitto_sub
```

Now enter each of the parameters on their own line:

```
-h HOSTNAME
-p 8883
--cafile /home/pi/containers/certificates/rootCA.pem
-u home
-P PASSWORD
```

Note:
Make sure you use the full path to the root CA file: `/home/pi/containers/certificates/rootCA.pem` instead of `containers/certificates/rootCA.pem`. If you use the latter, the mosquitto_sub command will only work when you're in your home directory.

After this, you can just execute the `mosquitto_sub` command as if you would do with an unsecured setup:

```
mosquitto_sub -t '#' -v
```

The connection works now because `mosquitto_sub` reads all the extra options from its configuration file. If you also want this to happen with its counterpart `mosquitto_pub`, just create the following symbolic link:

```
cd ~/.config
ln -s mosquitto_sub mosquitto_pub
```

This creates a symbolic link `mosquitto_pub` linking to the file `mosquitto_sub`, both in the `~/.config` directory. The `mosquitto_pub` command reads its default configuration from the file `~/.config/mosquitto_pub`, which is the same file as the `~/.config/mosquitto_sub` file that you created with nano.

In the rest of this book, I use the short commands for `mosquitto_pub` and `mosquitto_sub`. You can do the same if you put all the connection parameters in this configuration file.

4.3 ● Using graphical MQTT clients

You can now send messages to your MQTT broker and receive messages with a command line MQTT client. While the Mosquitto clients are useful for quick tests of your MQTT setup, you don't get a good overview of what is happening on your MQTT broker. So let's look at some graphical clients.

These clients can run on your home automation gateway or another Raspberry Pi or computer, as long as they have access to your gateway in your local network.

4.3.1 ● MQTT.fx

A nice graphical client is MQTT.fx (https://mqttfx.jensd.de), which is written in Java and runs on all major operating systems. You can find pre-compiled binaries for Windows, macOS, and Linux on its website.

When you run MQTT.fx for the first time, add a connection profile for your MQTT broker. Click on **Extras > Edit Connection Profiles** and then click on the plus sign at the bottom left.

Enter the following fields:

Profile Name
The name of your MQTT broker profile, for instance, **Home automation gateway**.

Profile Type
Leave this on **MQTT Broker**.

Broker Address
The hostname or IP address of your Raspberry Pi that is running the MQTT broker. If you use TLS, make sure that you enter the same hostname as in the certificate of the MQTT broker.

Broker Port
Leave this on **1883** for an unencrypted connection or change this to **8883** for an encrypted connection.

If you have enabled authentication on your MQTT broker, fill out the right fields in the **User Credentials** section. If you have enabled encryption, go to the **SSL/TLS** section, check **Enable SSL/TLS**, choose **CA certificate file**, and then select the `rootCA.pem` file you copied from your Raspberry Pi to your computer (see Chapter 3).

When you're finished with the configuration, click on **OK** to save and use this connection profile.

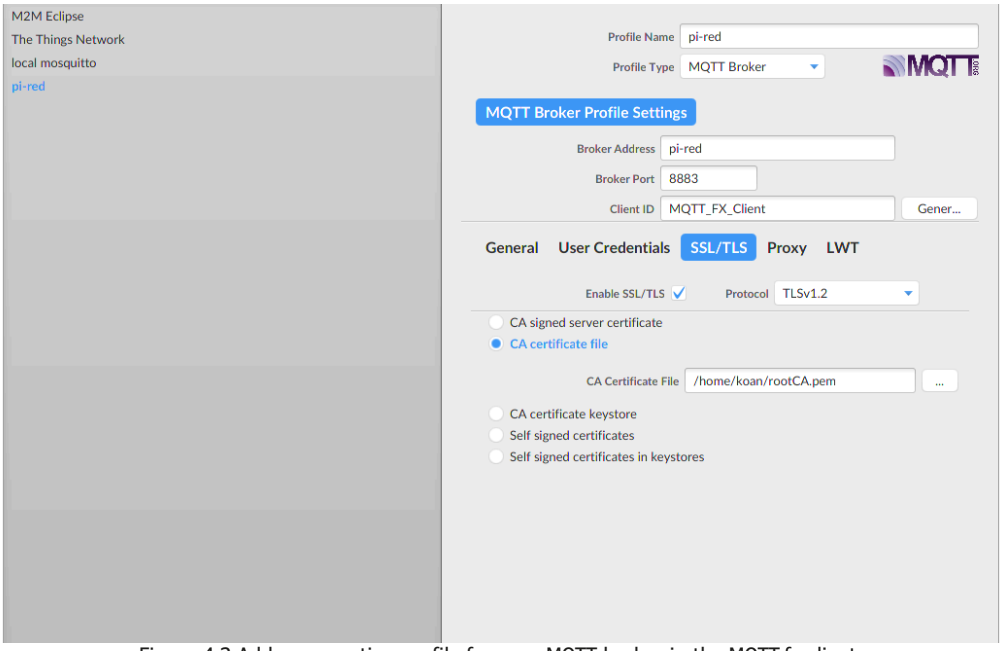

Figure 4.2 Add a connection profile for your MQTT broker in the MQTT.fx client.

With your newly added connection profile selected at the top left, click on the **Connect** button. If the connection settings are correct, the circle at the top right should get a green colour.

Now go to the **Subscribe** tab, enter an MQTT topic (wildcards such as # are allowed), and click on **Subscribe**. In the right panel, you'll see MQTT messages appearing as soon as they are sent to the broker. If you click on one of them, you can see its content. You can even decode the content as specific formats, such as JSON, Base64, or hexadecimal values.

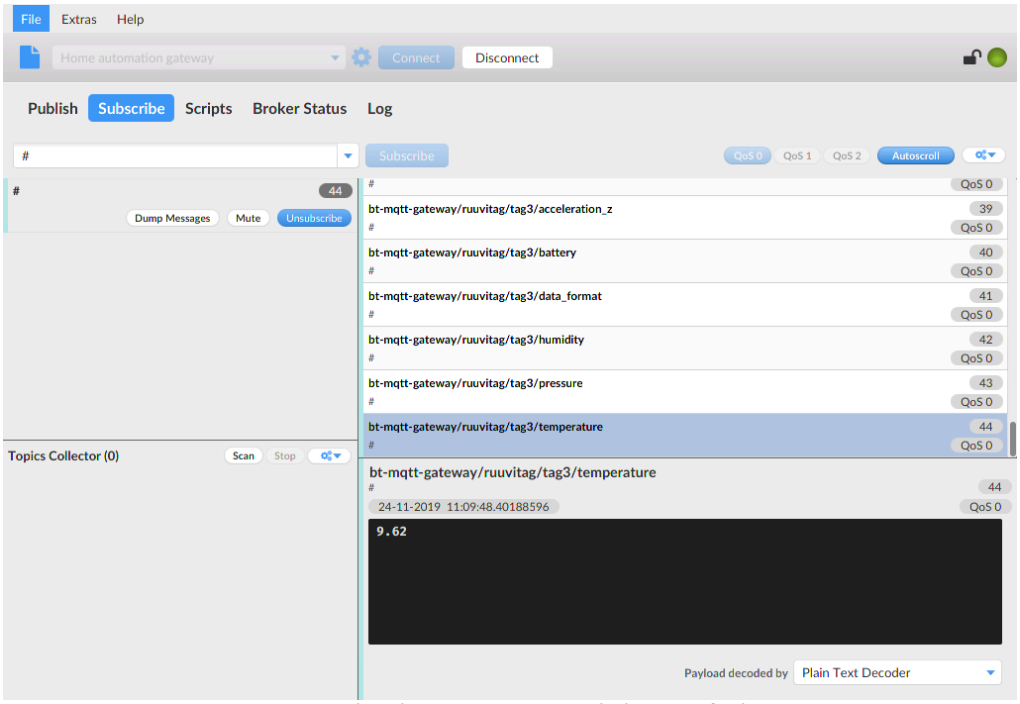

Figure 4.3 Subscribe to MQTT topics with the MQTT.fx client.

Publishing an MQTT message in MQTT.fx is equally easy: go to the Publish tab, enter the topic at the top left, enter the payload in the text field below and then click on Publish. If you switch to the Subscribe tab and you have subscribed to this topic, you'll see that it appears in the messages to the MQTT broker.

4.3.2 ● MQTT Explorer

Another graphical MQTT client, which is also available on Windows, macOS, and Linux, is MQTT Explorer (http://mqtt-explorer.com). It differentiates itself from other MQTT clients by providing a hierarchical overview of your MQTT topics.

When the program starts, it shows the available MQTT brokers to connect to. Click on the yellow plus sign at the top left to add a new broker. Then enter the following fields:

Name
The name of your MQTT broker profile, for instance, **Home automation gateway**.
Protocol
Leave this on **mqtt://**.

Host
The hostname or IP address of your Raspberry Pi that is running the MQTT broker. If you use TLS, make sure that you enter the same hostname as in the certificate of the MQTT broker.

Port
Leave this on **1883** for an unencrypted connection or change this to **8883** for an encrypted connection.

If you have enabled authentication and/or encryption on your MQTT broker, fill out the fields **Username** and **Password** and **enable Encryption (tls)**. Then click on **Advanced, Certificates** and **Server certificate (CA)**. Select the `rootCA.pem` file you copied from your Raspberry Pi to your computer (see Chapter 3).

When you're finished with the configuration, click two times on **Back** to return to the main profile window, then **Save** to save this connection profile and then **Connect** to use it.

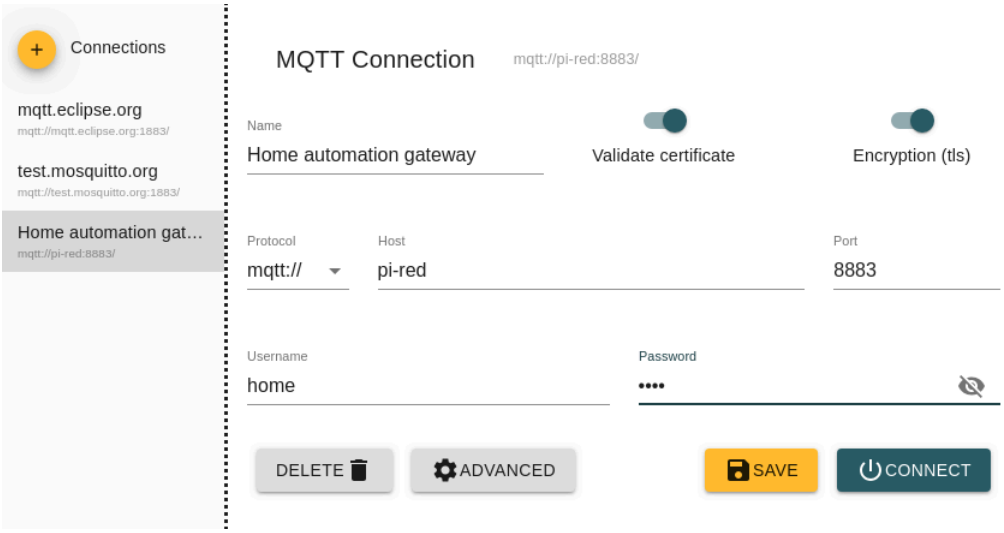

Figure 4.4 Add a connection profile for your MQTT broker in MQTT Explorer.

If the connection settings are correct, the icon at the top right should be green. In the left panel, you see a hierarchical view of the MQTT topics that have messages on the broker. By default, MQTT Explorer subscribes to all topics. Click on the small arrows to open topics and navigate the hierarchy.

Message payloads are shown in the right panel, and if the payload is a number, you can even see a graph if you click on the arrow next to **History**. Publishing an MQTT message can be done by entering the topic and payload at the bottom and then clicking on **Publish**.

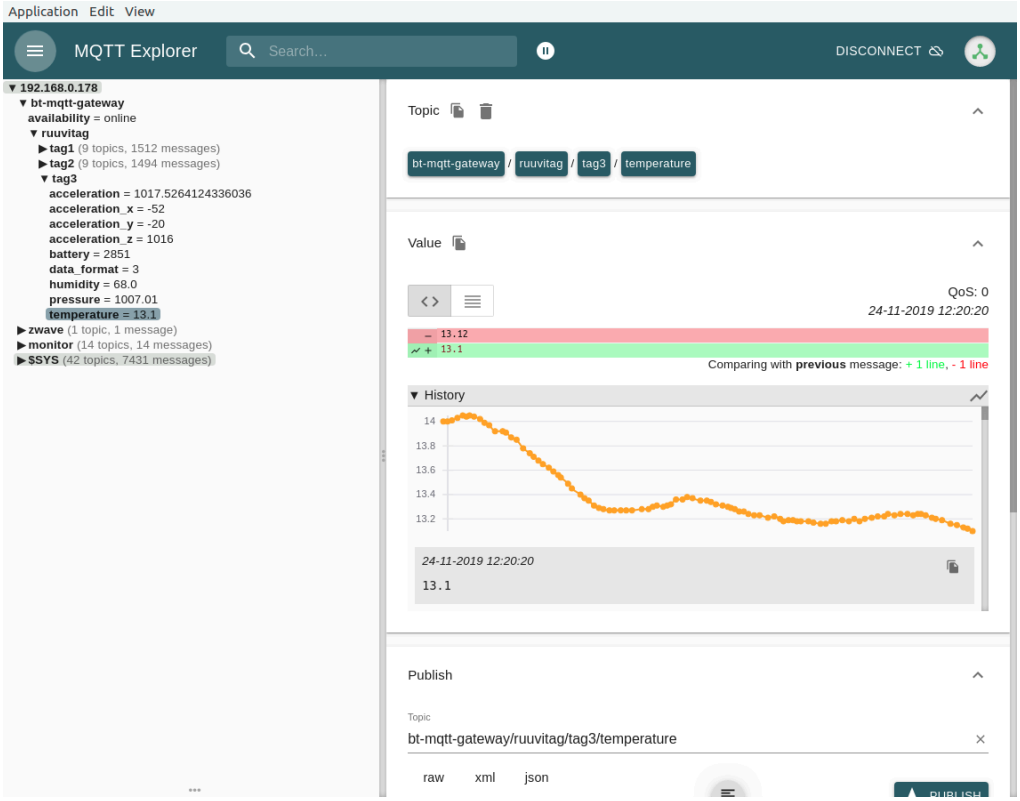

Figure 4.5 With a hierarchical view and graphs to interpret numerical payloads, MQTT Explorer is a very useful MQTT client.

4.4 ● Using MQTT in Python

If you want to process MQTT messages in your Python programs, you can use the Python library of the Eclipse Paho project (https://www.eclipse.org/paho/). This project implements an MQTT client in Python. As it's on PyPI (https://pypi.org/project/paho-mqtt/), you can easily install the library with pip, and the PyPI project page shows extensive documentation. You'll be using MQTT directly and indirectly in many Python programs in the rest of this book, so I'll limit myself here to a simple example: a client that lets you know the current time. Here's the code:

```
"""Send an MQTT message with the time to your broker when asked.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
from datetime import datetime

import paho.mqtt.client as mqtt

MQTT_HOST = "HOSTNAME"
MQTT_PORT = 8883
MQTT_CAFILE = "/path/to/rootCA.pem"
MQTT_USERNAME = "home"
MQTT_PASSWORD = "PASSWORD"
MQTT_CLIENT_ID = "Time"

MQTT_TOPIC_TIME_REQUEST = "time/request"
MQTT_TOPIC_TIME_REPLY = "time/reply"

TIME_FORMAT = "%Y-%m-%d %H:%M"


def on_connect(client, userdata, flags, rc):
    """Subscribe to the right MQTT topics after connecting."""
    print("Connected with result code " + str(rc))
    client.subscribe(MQTT_TOPIC_TIME_REQUEST)


def on_message(client, userdata, message):
    """Reply with the time when asked."""
    now = datetime.now().strftime(TIME_FORMAT)
    client.publish(MQTT_TOPIC_TIME_REPLY, now)


if __name__ == "__main__":
    # Initialize MQTT connection
    mqtt_client = mqtt.Client(MQTT_CLIENT_ID)
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    # Set up authentication and TLS encryption
    mqtt_client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)
    mqtt_client.tls_set(ca_certs=MQTT_CAFILE)
```

```
    # Connect and start event loop
    mqtt_client.connect(MQTT_HOST, MQTT_PORT)
    mqtt_client.loop_forever()
```

Most of the code is bookkeeping. In the beginning, I import some classes in the `datetime` and `paho.mqtt.client` modules. Then I define some constants with the MQTT host, port, location of the CA file, username, password, client ID, the used topics, and a format string for the time.

Then the `on_connect` function runs when the MQTT client is connected to the broker, because of the line `mqtt_client.on_connect = on_connect` at the end, which registers the `on_connect` function as a callback. In this function, the client subscribes to the topic `time/request`. That way every time someone publishes a message on the topic `time/request` on this MQTT broker, the client gets notified.

It's the `on_message` function that gets called when the client gets notified of a message because it's registered as a callback in the line `mqtt_client.on_message = on_message` at the end. The function doesn't have to check for which topic it's called, because the program has only subscribed to one topic. So the client then publishes the current date and time as a payload to the topic `time/reply`.

At the end of the program, the MQTT client object is created, the callbacks are registered, the username and password are supplied, the CA file for the TLS connection is given and the client connects to the MQTT broker. After this, the program starts an infinite loop, waiting for messages and then running the `on_message` callback.

Note:
If you want to connect to an unencrypted MQTT broker, just change the value of `MQTT_PORT` to `1883` and remove the call to `mqtt_client.tls_set`.

Also add a `requirements.txt` file with this content, a dependency of the program:

```
paho-mqtt
```

Set up a virtual environment and install the dependencies:

```
python3 -m venv .venv
source .venv/bin/activate
pip3 install -r requirements.txt
```

Optionally change the constants at the beginning of the `mqtt_time.py` file, and then run the program like this:

```
python3 mqtt_time.py
```

Now open another shell and subscribe to the `time/reply` topic:

```
mosquitto_sub -t 'time/reply' -v
```

In yet another shell, send a request to the program:

```
mosquitto_pub -t 'time/request' -m ''
```

The `-m ''` means that the message payload is empty, as the program just ignores the payload. If all goes well, you'll see the current date and time appearing in the shell running `mosquitto_sub`. If your test is completed, just quit the `mqtt_time.py` program with Ctrl+c.

This is just a simple example, but it shows the basic architecture of many MQTT programs: setting up an MQTT client, registering callbacks, connecting to the MQTT broker, starting the event loop, and then replying to the messages received.

Asking for the time is of course no home automation yet, but I'll show you a lot of examples in the rest of this book using the same approach.

4.5 ● Direct communication between other containers and Mosquitto

In the rest of this book, I'll add other Docker containers for various home automation services. Many of them are using MQTT as their communication protocol. As long as they are all running on the same Raspberry Pi as the Mosquitto container, it doesn't make much sense to let them communicate over TLS.

Luckily, Docker Compose creates an 'internal' network for all services you define in your `docker-compose.yml` file. Each container can reach other containers by their internal hostname, which is identical to the `container_name` defined for the service in the Docker Compose file. So if another container wants to connect to the MQTT broker running in the Mosquitto container, it can do this using the hostname `mosquitto`, and none of this communication leaves the Raspberry Pi: it's all going through a virtual network.

This way all communication from and to other devices with the MQTT broker running on your Raspberry Pi will happen encrypted over TLS on port 8883 (or 9091 with WebSocket

over TLS), while all communication from and to other containers can happen unencrypted over the internal network using the default port 1883.

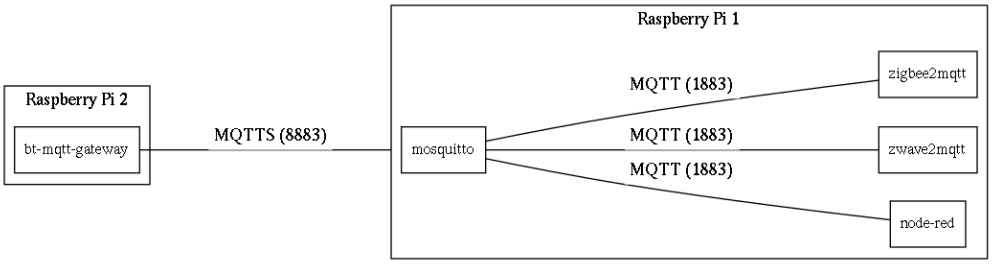

Figure 4.6 The Docker containers on one Raspberry Pi communicate with each other unencrypted on the internal Docker network, while the container on the second Raspberry Pi communicates over an encrypted connection.

If you execute the `docker ps` command, you already see this in the **PORTS** column of the mosquitto container. This shows **0.0.0.0:8883->8883/tcp**, **1883/tcp, 0.0.0.0:9091->9091/tcp**. What this means is that TCP ports 8883 and 9091 are forwarded to the same externally reachable ports of your Raspberry Pi, while TCP port 1883 (which isn't shown with **0.0.0.0:1883->** before it) is not externally reachable. But for containers in the same internal network created by Docker Compose, it is reachable.

However, the fact that port 1883 is exposed to the other containers doesn't mean that something is listening on the port. In the secure configuration of Mosquitto that I showed earlier in this chapter, the MQTT broker only listens on ports 8883 and 9091. So you should change Mosquitto's configuration so it also listens on port 1883:[2]

```
# Unencrypted MQTT
port 1883

# MQTT over TLS
listener 8883
cafile /mosquitto/config/certs/rootCA.pem
keyfile /mosquitto/config/certs/key.pem
certfile /mosquitto/config/certs/cert.pem

# MQTT over WebSocket over TLS
listener 9091
protocol websockets
cafile /mosquitto/config/certs/rootCA.pem
```

2 If you want to add unencrypted MQTT over WebSocket for other Docker containers, just add the lines listener 9001 and protocol websocket to the first section in Mosquitto's configuration file.

```
keyfile /mosquitto/config/certs/key.pem
certfile /mosquitto/config/certs/cert.pem

# Authentication and access control
password_file /mosquitto/config/passwords
allow_anonymous false
acl_file /mosquitto/config/acl

# Miscellaneous
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
```

In the next chapters, you'll keep adding services to the docker-compose.yml file in your home directory, so it looks something like this:

```
version: '3.7'

services:
  mosquitto:
    image: eclipse-mosquitto
    container_name: mosquitto
    restart: always
    ports:
      - "8883:8883"
      - "9091:9091"
    volumes:
      - ./containers/mosquitto/config:/mosquitto/config
      - ./containers/mosquitto/data:/mosquitto/data
      - ./containers/mosquitto/log:/mosquitto/log
      - ./containers/certificates:/mosquitto/config/certs:ro
      - /etc/localtime:/etc/localtime:ro
    user: "1000:1000"
  motioneye:
    # motionEye configuration
  bt-mqtt-gateway:
    # bt-mqtt-gateway configuration
```

Your Docker Compose file will always have the mosquitto service, because this is central to the architecture used in this book. The lines starting with the # for the other services are comments. These are not configuration values, but some remarks you can add to explain things to yourself or others. I have used them here just as placeholders. You'll see in the rest of the book which configuration you should add to these services.

In the other chapters, I'll use such a placeholder for the configuration of the `mosquitto` service, so I don't have to repeat this every time in the Docker Compose file. I also don't repeat other services from previous chapters. So you should keep in mind that I only show a part of the Docker Compose file each time I explain a new service. At the end of this book, you should have a big Docker Compose file containing all your home automation services.

Note:
For some services, it could be interesting to run them on another Raspberry Pi to your main home automation gateway. For these services, your Docker Compose file shouldn't contain the `mosquitto` service, and the communication to the MQTT broker should happen over TLS using the hostname of your home automation gateway. However, if the service doesn't support MQTT over TLS, you do need to run mosquitto on the machine and configure it as a bridge to communicate with your main MQTT broker over TLS. See the appendix at the end of this book for details.

4.6 ● Summary and further exploration

This chapter is really about the core architecture of this book. I introduced the lightweight network protocol MQTT that is used for almost every software in this book. You learned about MQTT brokers, publishers and subscribers, messages, topics, hierarchical names, and wildcards.

I also guided you through the installation and configuration of Mosquitto, including a secure setup with TLS (which gives you server authentication and encryption), client authentication with a password, and an access control list. Afterwards I showed you some graphical clients that come in handy to test your MQTT setup, and you learned how to subscribe to MQTT topics and publish MQTT messages in Python using the Paho MQTT library.

MQTT is a very interesting protocol. If you want to know more about it, consult the documentation on the official website (https://mqtt.org). Eclipse Mosquitto also has a lot of documentation on its website (https://mosquitto.org), including the excellent man pages, such as the one for `mosquitto.conf` (https://mosquitto.org/man/mosquitto-conf-5.html). These are essential resources if you need a more complex broker configuration.

In the rest of this book, I introduce various services that translate other home automation protocols to MQTT messages. There are also Wi-Fi devices that support MQTT directly, and it's interesting to explore these.

For instance, the Shelly devices I cover in the next chapter offer an MQTT API (https://shelly-api-docs.shelly.cloud/#mqtt-support), although unfortunately not with TLS. You could also buy the inexpensive Sonoff devices (https://sonoff.tech) and flash them with the open-source firmware Tasmota (https://tasmota.github.io/docs/), which supports MQTT (although you have to build the firmware yourself to get TLS support).

MQTT is also lightweight enough that you can use it in devices you create yourself. The ESP8266 and ESP32 are popular choices for DIY Wi-Fi devices for home automation. The PubSubClient library (https://pubsubclient.knolleary.net) for Arduino can be used on both devices, although TLS support is not built-in.[3]

3 If you want to learn how to create your own home automation devices with the ESP8266 and MQTT, I can recommend the book "IoT Home Hacks with ESP8266" by Hans Henrik Skovgaard (https://www.elektor.com/iot-home-hacks-with-esp8266-e-book).

Chapter 5 ● TCP/IP

Many home automation products use specialized protocols, mostly wireless, that require you to have a specific transceiver. However, many other products just use the network protocols that you're used to on your desktop computer, laptop, smartphone, and tablet. This is the internet protocol suite, commonly known as TCP/IP because the fundamental protocols in this suite are the Transmission Control Protocol (TCP) and the Internet Protocol (IP).

The internet protocol suite consists of protocols in four layers (from top to bottom):[1]

Application layer
Application protocols operate in this layer: they let applications communicate to other applications. MQTT (see Chapter 4) is one example of a protocol in the application layer, as are SSH, HTTP, and DNS.

Transport layer
Transport protocols provide a communication channel between hosts on the network. TCP and UDP are the dominant transport layer protocols.

Internet layer
Protocols in this layer interconnect networks and hence create the 'internet'. IP (IPv4 and IPv6) is the internet layer protocol you'll use the most.

Link layer
Protocols in this layer transmit network packets over a physical medium. Examples are ARP and NDP.

The big advantage of using TCP/IP for home automation is that you don't need any specific hardware. The Raspberry Pi has already Wi-Fi and Ethernet, and your other computers and IP cameras too. You can even find many Wi-Fi-enabled home automation products, such as smart lights, smart switches, and so on. So without having to buy any specific bridges, transceivers, or gateways, you can already have a good taste of home automation.

In the previous chapter you encountered MQTT. In this chapter, I'll show you some other possibilities to automate your home with only a network connection.

5.1 ● Wake other network devices

If you have computers on your network attached using an Ethernet interface, you can make use of the Wake-on-LAN (WoL) standard.[2] This is done by sending a "magic packet" to all computers on your network. This packet contains the MAC address of the network

1 This is the canonical naming of the four layers as defined by RFC 1122 (https://tools.ietf.org/html/rfc1122). Other network models add layers or give the lower layers other names. For instance, some network models split the link layer into a data link layer and a physical layer.
2 Wake-on-LAN is quite low-level: it works in the link layer of the internet protocol suite.

interface of the computer you want to wake. A computer that supports WoL will have his network interface listening to these incoming packets. If it recognizes its MAC address, it signals the computer's power supply to boot the system, as if you're physically pressing the power button. This not only works for computers but also printers, NAS systems (network-attached storage), and so on.

Wake-on-LAN is only able to wake the computer because (part of) the network interface card is still on. This consumes a very small amount of standby power. If you physically remove the device's power plug, of course, the network interface card isn't able to do anything anymore, so you can't wake the device.

Note:
Wake-on-LAN only works for ethernet devices. For Wi-Fi devices there's a Wake on Wireless LAN (WoWLAN) standard, but it's not widely implemented and has several limitations which don't make it very practical except in some very specific cases.

If you want to wake another device with WoL from your Raspberry Pi, first install the `wakeonlan` package:

```
sudo apt install wakeonlan
```

Then wake the device with:

```
wakeonlan bc:30:5b:de:ee:94
```

Of course, substitute the MAC address for the MAC address of your device. If you don't know the device's MAC address, you can find it in the list with DHCP leases of your home router.

To wake one device, this `wakeonlan` command is easy, but if you want to wake more devices, you should use a more user-friendly approach, such as Home Assistant's Wake on LAN component (https://www.home-assistant.io/integrations/wake_on_lan/). See Chapter 10 for more information about Home Assistant.

Note:
You can wake other devices from your Raspberry Pi with Wake-on-LAN, but you can't wake your Raspberry Pi from other devices: the Raspberry Pi's ethernet port doesn't have the necessary circuitry.

5.2 ● Remote control with SSH

Waking other devices on your network is neat, but what about shutting down other devices? This is possible too. Even more: you can run arbitrary commands on remote devices on your network. You already know how to do this: with the `ssh` command (see Chapter 2).

So the only thing you need is an OpenSSH server running on the device. Most Linux machines have this running or installed by default, or just one command away from installation. This includes:

- a Raspberry Pi running Raspberry Pi OS or another Linux distribution;
- your wireless router or access point (especially if it's running OpenWrt);
- your NAS (network-attached storage) system.

If not, consult the operating system's manual to discover how to install and enable the OpenSSH server.[3]

OpenSSH is not only for Linux: it also runs on macOS (which shares the same UNIX ancestry as Linux), and recently it has even become possible to run an OpenSSH server on Windows 10. So it's the ideal mechanism to run commands on other devices and even to shut them down remotely.

Note:
To install the OpenSSH server in Windows, revisit the installation instructions for the OpenSSH client in Chapter 2, but this time choose OpenSSH Server. Then after installation, start PowerShell as an administrator, run `Start-Service sshd` to start the server, and `Set-Service -Name sshd -StartupType 'Automatic'` to configure it to start automatically every time Windows boots.

Now you can just log in to the devices with the SSH client on your Raspberry Pi:

```
ssh USERNAME@ADDRESS
```

Substitute `USERNAME` and `ADDRESS` by the username you have on the device and the IP address or hostname the device has on your network. See Chapter 2 for more information about the login progress.

After you have entered the correct password for the username on the device, you get a login prompt. This can look a bit different from what you're used to on your Raspberry Pi,

3 On a Debian-based Linux distribution with systemd, you enable and start the OpenSSH server with `sudo systemctl enable --now ssh`. By enabling the service it starts automatically every time Raspberry Pi OS boots.

but usually, it has the same components: your username, the device's hostname, and your current directory.

5.2.1 Run commands on other devices

Now you can enter commands on the prompt as if you're sitting at the device with a keyboard and monitor. But you're doing this for home automation purposes, so you generally don't want to interactively type commands. So log out with exit.

Now log in again, but this time with a command at the end of your command line:

```
ssh USERNAME@ADDRESS df -h
```

After you have entered your password, the `ssh` command logs into the device, runs the `df -h` command (which shows the device's disk usage) and logs out.[4]

So if you want to turn off the device, you're going to use this command if it's a Linux device:

```
ssh USERNAME@ADDRESS sudo shutdown -h now
```

This assumes that the user is non-privileged but has been given the rights by sudo to run the shutdown command without having to enter a password. Raspberry Pi OS already has this configuration (see Chapter 3). If this is not your configuration, run the following command on the device:[5]

```
sudo visudo
```

Then add the following configuration lines:

```
# Allow members of group sudo to execute any command
%sudo ALL=(ALL) NOPASSWD: ALL
```

This means that every user that belongs to the sudo group can run all commands without a password.

4 Of course `df` is a Linux/UNIX command and won't work when you use this on a Windows machine.

5 I assume that you're running these commands as a non-privileged user with sudo rights. If you're logged in as the `root` user on the device, run the commands without `sudo`.

After saving the file with Ctrl+o and quitting nano, add the user to the sudo group with:

```
sudo adduser USERNAME sudo
```

You can also turn off a Windows computer remotely:

```
ssh USERNAME@ADDRESS shutdown /h /f
```

This assumes that the Windows user is an Administrator, who has the right to turn off the computer.

5.2.2 ● Secure passwordless logins using SSH keys

Now you can remotely run a command, for instance, to turn off a device, but you still have to enter your password while logging in on the device. This means that you can't use this `ssh` command in scripts to automate remote tasks.

Luckily SSH has another way to log in: public-key authentication. First, create a key pair on your Raspberry Pi:

```
ssh-keygen
```

Press **ENTER** for the default file path and two times **ENTER** again for an empty passphrase and its confirmation.[6] After this, you see a fingerprint and 'randomart image' for your key pair. If you have changed nothing to the default values, your public key is saved in `/home/pi/.ssh/id_rsa.pub` and your private key in `/home/pi/.ssh/id_rsa`.

6 Choosing an empty passphrase is not the most secure way of working with key pairs, but it's acceptable on a local network. If you want to have a secure key pair but don't want to enter a passphrase every time you log in with the key pair, you can use `ssh-agent`: it asks you for the passphrase the first time and then remembers it the following times.

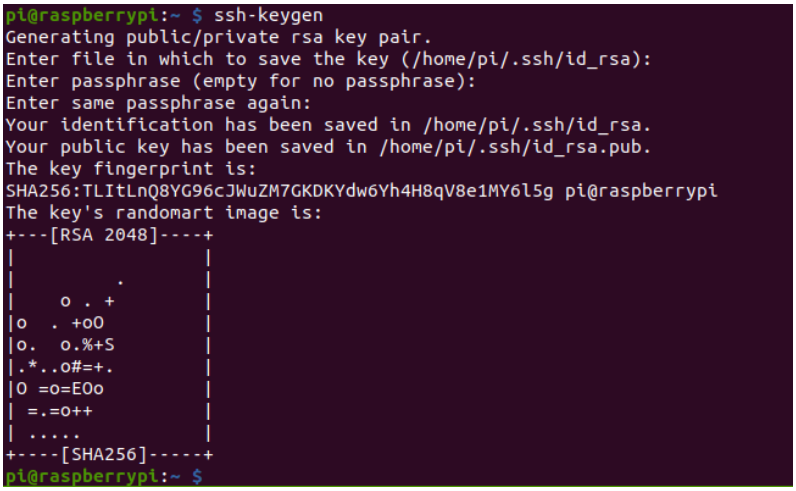

Figure 5.1 Create a key pair for passwordless logins using SSH.

Now you have to add your public key to the remote system's database of trusted keys. If the remote system is a Linux system (such as Raspberry Pi OS), this is simple:

```
ssh-copy-id USERNAME@ADDRESS
```

If you have changed the default file name, use:

```
ssh-copy-id -i ~/.ssh/name_of_file.pub USERNAME@ADDRESS
```

The `ssh-copy-id` command logs into the other computer (you have to enter your password this time) and then saves your public key in the database of trusted keys.

If the target system is Windows, the procedure is a bit more laborious. First show the content of your public key on your Raspberry Pi:

```
cat ~/.ssh/id_rsa.pub
```

The output should start with `ssh-rsa` and then a long list of characters. Copy the whole output, including the `ssh-rsa` part.

Then log in to your account on Windows, open Explorer and navigate to `C:\Users\USERNAME\.ssh`, with USERNAME your username on Windows. If there's no `.ssh` directory in your user directory, create it. Enter the .ssh directory, create a new file: `authorized_`

`keys` there (note: this file doesn't have a file extension), and open it in Notepad. Paste the content of the `id_rsa.pub` file from your Raspberry Pi in it and save the file.

The next step is to set the permission of this file right. So right-click on the `authorized_keys` file, go to **Properties**, then **Security** and then **Advanced** and click **Disable Inheritance**. Also, choose **Convert inherited permissions** into **explicit permissions on this object.** Then remove all permissions, except for **SYSTEM** and your username, who should have **Full Control**.

After this, every ssh command you enter on your Raspberry Pi to log into the computer and run commands, will use your private key to log in. The computer has your corresponding public key and can verify with it that you're using the private key to log in. So now you have passwordless logins to a remote computer, and you can automate running remote commands with SSH.

Warning:
Anyone with access to your Raspberry Pi can now log in to any remote system that trusts your public key, without having to enter a password. So it's very important that you secure your Raspberry Pi (see Chapter 3).

5.3 ● Collecting information from devices using SNMP

SNMP (Simple Network Management Protocol) is an application layer protocol to collect information from devices on your network and configure settings. Typical devices that support SNMP are printers, cable modems, switches, routers, and so on. SNMP is used to poll information such as up/down status, interface utilization, and ink level.

I'm not going to explain configuration changes through SNMP in this book but will show you how you can collect usable information from some of your devices. As an example, I'll collect information from:

OPNsense/pfSense
open-source router operating systems based on FreeBSD

The HP Color LaserJet Pro MFP M281fdw
an all-in-one wireless color laser printer

5.3.1 ● Walking through the MIB tree

All data that can be accessed using SNMP is described through a Management Information Base (MIB), structured in a hierarchical tree. So if you want to know which information you can collect from a device, you have to know which MIB it's using. In this MIB each data object has an Object Identifier (OID).

First, install an SNMP client and a package with many widely used MIBs:

```
sudo apt install snmp snmp-mibs-downloader
```

Now, configuring SNMP on the device you want to monitor is out of scope for this book, but you need to know two parameters:

The SNMP version
This can be 1, 2c, or 3. The latter is more secure, but isn't used that much. Most devices you'll encounter use version 2c.

The SNMP community
This is something like a group name. Most devices use public by default.
If you know that a specific device on your network has SNMP configured and is using SNMP version 2c with community string public, you can get a list of all available SNMP OIDs with:

```
snmpwalk -v2c -c public IP
```

Substitute `IP` by the IP address or hostname of the device.

Now you'll see a lot of OIDs and their values scrolling by. But if you want to get some more human-readable names, add the `-m ALL` option:

```
snmpwalk -v2c -c public -m ALL IP
```

You can go through these and see which ones are interesting to collect.

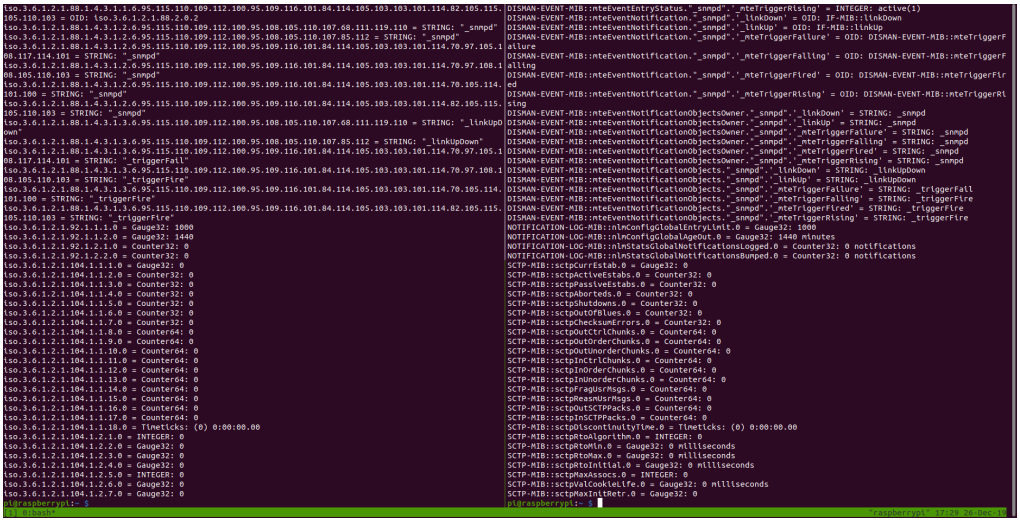

Figure 5.2 Translating the OIDs (left) to their human-friendly names (right) with the `-m ALL` option gives a lot more information if you want to collect data using SNMP.

Warning:
SNMP is not a very secure protocol. If you enable SNMP version 1 or 2c on a device (and typically it's enabled by default), everyone on your network can collect information of the device using snmpwalk or a similar command. The minimal security measure you should take is to change the community string to something else than public. But if you're serious about security, use SNMP 3 with authentication and encryption (which is the beyond scope of this book), or disable SNMP and use a more secure protocol.

5.3.2 ● Collecting your router's version using SNMP

If you know the OID of some data you're interested in, collecting these data from the device is as easy as:

```
snmpget -v2c -c public 192.168.0.1 SNMPv2-MIB::sysDescr.0
```

This example shows you the system description of the OPNsense router on IP address 192.168.0.1. This shows something like:

```
SNMPv2-MIB::sysDescr.0 = STRING: FreeBSD OPNsense.home 11.2-RELEASE-p10-
HBSD FreeBSD 11.2-RELEASE-p10-HBSD  5e5adf26fc3(stable/19.1) amd64
```

Now if you want to get the system's version, you can use the `grep` command on this output to only show the string after "`FreeBSD`" and then tail to only show the last time it matches.

This gives the following command:

```
snmpget -v2c -c public 192.168.0.1 SNMPv2-MIB::sysDescr.0|grep -o ⏎
"FreeBSD \S*"|tail -n 1
```

This gives (at the moment of writing) as a result:

```
FreeBSD 11.2-RELEASE-p10-HBSD
```

If you're doing an `snmpwalk`, you'll see a lot of other interesting data you can get from your router, including network interface statistics, IP to MAC address mappings, memory and storage usage, and so on.

5.3.3 ● Collecting your printer's ink levels

If you're doing an `snmpwalk` on your printer's IP address, chances are that you'll see a lot of interesting information scrolling by. For instance, you can ask for the printer's status:

```
snmpget -v2c -c public 192.168.0.104 HOST-RESOURCES-MIB::hrPrinterStatus.1
```

You'll get output like this:

```
HOST-RESOURCES-MIB::hrPrinterStatus.1 = INTEGER: idle(3)
```

You see that the result is an integer value, 3, but `snmpget` is friendly enough to translate this to a human-friendly value, "idle". If you want to know which other possible values exist, you'll have to look at the specification of HOST-RESOURCES-MIB. You can find this on http://www.net-snmp.org/docs/mibs/. Look for the name of the MIB and click on the link. There you'll see a table with the possible values for `hrPrinterStatus`:

Value	Printer status
1	other
2	unknown
3	idle
4	printing
5	warmup

Table 5.1 Values for hrPrinterStatus

If you want to know how long your printer has been on, this is also possible:

```
snmpget -v2c -c public 192.168.0.104 DISMAN-EVENT-MIB::sysUpTimeInstance
```

This shows something like:

```
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (265804) 0:44:18.04
```

Now if you want to parse this, you'll have to use some options for the `snmpget` command:[7]

```
snmpget -v2c -c public 192.168.0.104 -Ov -OQ DISMAN-EVENT-⏎
MIB::sysUpTimeInstance
```

The `-Ov` option removes the **DISMAN-EVENT-MIB::sysUpTimeInstance** = and the `-OQ` option removes the type information (**Timeticks**), so you get only the duration:

```
0:44:18.04
```

If you use `-Ot` instead of `-OQ`, you get the duration in Timeticks, which are hundredths of a second. To convert to minutes, divide this number by 6000. Now you can use this result for instance to notify you when your printer has been on for longer than half an hour.

But now the most important information for a printer: the ink levels. If you search around the output of `snmpwalk` for your printer, you'll easily find how to get the printer's ink levels. On my printer, I get the following relevant OIDs:

7 Use `man snmpcmd` to see all the available options.

```
Printer-MIB::prtMarkerSuppliesDescription.1.1 = STRING: "Black Cartridge HP
CF540A"
Printer-MIB::prtMarkerSuppliesDescription.1.2 = STRING: "Cyan Cartridge HP
CF541A"
Printer-MIB::prtMarkerSuppliesDescription.1.3 = STRING: "Magenta Cartridge
HP CF543A"
Printer-MIB::prtMarkerSuppliesDescription.1.4 = STRING: "Yellow Cartridge HP
CF542A"
Printer-MIB::prtMarkerSuppliesSupplyUnit.1.1 = INTEGER: percent(19)
Printer-MIB::prtMarkerSuppliesSupplyUnit.1.2 = INTEGER: percent(19)
Printer-MIB::prtMarkerSuppliesSupplyUnit.1.3 = INTEGER: percent(19)
Printer-MIB::prtMarkerSuppliesSupplyUnit.1.4 = INTEGER: percent(19)
Printer-MIB::prtMarkerSuppliesMaxCapacity.1.1 = INTEGER: 100
Printer-MIB::prtMarkerSuppliesMaxCapacity.1.2 = INTEGER: 100
Printer-MIB::prtMarkerSuppliesMaxCapacity.1.3 = INTEGER: 100
Printer-MIB::prtMarkerSuppliesMaxCapacity.1.4 = INTEGER: 100
Printer-MIB::prtMarkerSuppliesLevel.1.1 = INTEGER: 60
Printer-MIB::prtMarkerSuppliesLevel.1.2 = INTEGER: 42
Printer-MIB::prtMarkerSuppliesLevel.1.3 = INTEGER: 51
Printer-MIB::prtMarkerSuppliesLevel.1.4 = INTEGER: 55
```

Now how will you monitor or visualize all this information for your printer? You could write a shell script or Python script that periodically collects this information and reacts to it. But a more robust way is to use an SNMP integration in a home automation system such as Home Assistant (see Chapter 10). For instance, Home Assistant supports an SNMP OID as a sensor or a switch (https://www.home-assistant.io/integrations/snmp/). Many of these integrations don't support the human-readable OIDs but only their numerical counterparts. If you want to know which is the numerical representation of an OID, use the `-On` option (but without `-Ov`) in your `snmpget` command.

5.4 • Using devices with a HTTP/REST API

A more widely supported family of protocols is HTTP (Hypertext Transfer Protocol), HTTPS (HyperText Transfer Protocol Secure) and REST (Representational state transfer). These are the basic protocols of the web: they are not only used by every web site you visit, but many web-enabled devices support them too.

HTTP knows various methods, of which the most commonly used are GET and POST: the former is used to fetch a web page and the latter to submit a form. REST is essentially a set of useful conventions for structuring an API on top of HTTP, by redefining HTTP methods to do interesting stuff with other software than a web browser.

GET is used to get information about some object, while PUT is used to create some object. What this object looks like is up to the API's implementer. For instance, this can look like JSON (JavaScript Object Notation), a structured object. Which object you want to get

information about or create is set in the URL. For instance, the API could have a `/led/0` page for LED 0, `/led/1` for LED 1, and so on.

More and more home automation devices are working over Wi-Fi and are using an HTTP or REST API. Unfortunately, many of these only use a HTTP API with a web server in the cloud as an intermediary: you connect with a server of the manufacturer, using an API key as authentication, and you don't talk directly to your device, even if you and your device are both on your local network. As explained in Chapter 1, this is not desirable.

5.4.1 ● Setting up a Shelly device for secure remote control

A nice counterexample is the Shelly family of devices (https://shelly.cloud): while the manufacturer encourages you to use their cloud service, all their devices also support an HTTP API that works completely locally. That is, you're connecting directly to the device using its local IP address and all your API calls work completely disconnected from Shelly's servers.[8]

The details of the initial installation of the Shelly devices is beyond the scope of this book. But roughly it goes like this for the Shelly RGBW2 LED controller (see the figure below): the Shelly device is powered by a 12 V DC power adapter, and it powers a LED strip and controls the R, G, B, and W signal lines.

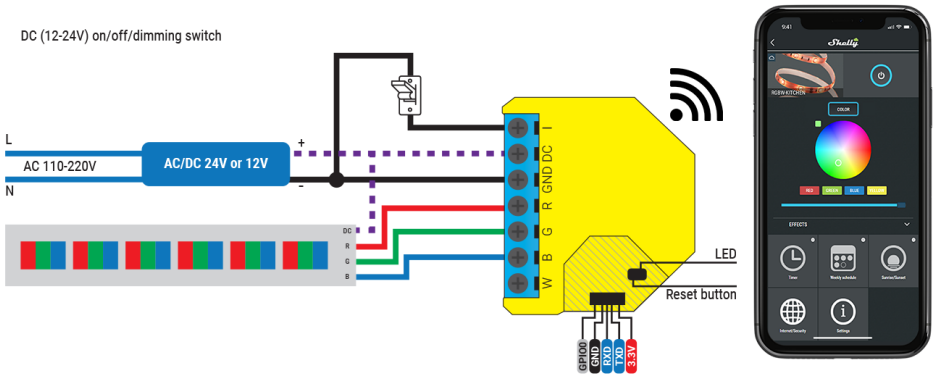

Figure 5.3 The Shelly RGBW2 LED controller is not only accessible with a mobile app, but also a web interface over Wi-Fi and a HTTP API.

When the Shelly device is powered on the first time, it starts an open Wi-Fi access point with the SSID **shelly<MODEL>-XXXXXXXXXXXX**. If you connect to it and then visit http://192.168.33.1 in your web browser, you get access to its web interface. In the

8 The Shelly devices also support MQTT, see Chapter 4.

Internet & Security settings, change the Wi-Fi mode to Client and enter your own Wi-Fi network's SSID and its password. After you save these settings, the device reboots and connects to your network.

Now find the new IP address of your Shelly device in your router's DHCP address leases. Visit this new IP address in your web browser, after which you see the same interface as before. First, return to the **Internet & Security** settings. Choose **Restrict Login** and enable the option so you need to enter a username and password to access the web interface and control the device. Change the default settings (**admin** and **admin**) and save them. After this, your Shelly device is ready to be controlled securely by the HTTP API.

Warning:
At the time of writing this book, the Shelly devices don't support HTTPS, so all communication (including the username and password you supply) is unencrypted.

5.4.2 ● Using Shelly's HTTP API with curl

The HTTP API for Shelly devices is neatly documented on https://shelly-api-docs.shelly.cloud/#common-http-api. If there are device-specific extensions to the API, these can be found in the device's section on that page.

I'll be using the `curl` command on the Raspberry Pi's command line to show how you use the API. First, here's how you get some basic information about your device:

```
curl http://192.168.0.202/shelly
```

This does not require you to enter a username or password, even if you have authentication enabled. But the only information you're getting anyway is the device type (such as "SHRGBW2"), the MAC address, whether authentication is enabled, the firmware version, and the number of outputs you can control (4 for the RGBW2 LED controller).

The information is returned on one line in a long string of characters in JSON format. If you want to have a better look at it in a structured way, pipe the `curl` command's output to the `jq` JSON processor:[9]

```
curl http://192.168.0.202/shelly | jq
```

Now if you read about the `/status` endpoint and try to get the status information, you'll

9 You can install jq with `sudo apt install jq`.

see that you get a "401 Unauthorized" error.

This is because the device requires authentication. So you have to add your username and password after the `-u` option separated by a colon (`:`):

```
curl -u USER:PASSWORD http://192.168.0.202/status | jq
```

This will give you a lot of information, including the availability of a firmware update and the current RGBW values of the LEDs.

What about controlling the LEDs? You can do this with the /color/0 endpoint. For instance, turn off the LEDs by setting the `turn` parameter to `off`:

```
curl -u USER:PASSWORD http://192.168.0.202/color/0?turn=off| jq
```

And turn them on again:

```
curl -u USER:PASSWORD http://192.168.0.202/color/0?turn=on | jq
```

Note that you still get JSON output: the command not only controls the LEDs but also returns the current state of the LEDs.

Note:
Have a look at Shelly's documentation for all other things you can control, such as effects (the `effect` parameter), individual R, G, B and W values, timers, and so on.

5.4.3 ● Using the HTTP API in Python

The `curl` command gives you a quick way to test a new device, but for serious home automation, you need a programming language such as Python or a full-blown home automation environment such as Home Assistant (see Chapter 10).

In this subsection, I'll show you how you use the Shelly's (or any other device's or service's) HTTP API in Python with the `requests` library. For instance, here's a simple Python script that first gets Shelly's status information and then continuously flashes the LEDs red with an interval of one second:

```
"""Let a Shelly RGBW2 LED controller flash the LEDs red.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
import time

import requests

# Define URLs for the Shelly HTTP API
IP = "http://192.168.0.202"
SHELLY = IP + "/shelly"
COLOR = IP + "/color/0/"
DISABLE_EFFECTS = "effect=0"
ONLY_RED = "red=255&green=0&blue=0&white=0"
ON = "turn=on"
OFF = "turn=off"
AUTH = ("admin", "admin")
HEADER = {"Content-Type": "application/x-www-form-urlencoded"}

try:
    shelly = requests.get(SHELLY)
    if shelly.status_code == 200:
        shelly_json = shelly.json()
        shelly_type = shelly_json["type"]
        shelly_mac = shelly_json["mac"]
        print(
            "Connected to device {} with MAC address {}.".format(
                shelly_type, shelly_mac
            )
        )

    # Disable effects and show only red
    disable_effects = requests.post(
        COLOR, auth=AUTH, headers=HEADER, data=DISABLE_EFFECTS
    )
   only_red = requests.post(COLOR, auth=AUTH, headers=HEADER, data=ONLY_RED)

    while True:
        on = requests.post(COLOR, auth=AUTH, headers=HEADER, data=ON)
        time.sleep(1)
        off = requests.post(COLOR, auth=AUTH, headers=HEADER, data=OFF)
        time.sleep(1)
except requests.exceptions.ConnectionError as e:
    print("Can't connect to device {}: {}".format(IP, e))
```

So this code first defines some constants for the URLs and authentication values. You have to change these to your situation. Then the code gets the "/shelly" URL (`requests.get` does a HTTP GET request) to show the device type and MAC address. This is done by checking whether the HTTP status code is 200 and extracting some JSON values from the returned data. From this JSON formatted content, the code needs the `type` and `mac` values.

For the next calls to the HTTP API, the code uses HTTP POST requests (with `requests.post`). All these HTTP requests need authentication, so that's why they have the `auth=AUTH` parameter.

According to Shelly's documentation, all parameters should be supplied either as a query string in the URL (the parameters are added after the ? character in the URL, like in the previous subsection on the command line) or as an application/x-www-form-urlencoded POST payload (https://shelly-api-docs.shelly.cloud/#http-dialect). In this Python code, the latter approach is used: that's why the `"Content-Type": "application/x-www-form-urlencoded"` header is added. The `data` parameter contains the parameter for the URL, for instance `effect=0` to disable effects and `red=255&green=0&blue=0&white=0` to only show the red LEDs.

So using the same approach for all these POST requests, the code disables effects, enables only the red LEDs, and then continuously turns these LEDs on and off for a second, thereby flashing the LED strip.

If you want to exit the program, press **Ctrl+c**.

There's a lot more about HTTP and REST APIs. If you're serious about writing Python programs to use the HTTP API of your devices, the documentation of the requests package (https://requests.readthedocs.io) has a lot of information about more advanced features.

Note:
There's a package called ShellyPy (https://pypi.org/project/ShellyPy/) which is a Python wrapper around the Shelly HTTP API. This lets you do things like `device = ShellyPy.Shelly("192.168.0.220")` and then `device.relay(0, turn=True)`. Unfortunately, Shelly RGBW2 isn't supported yet.

5.5 ● Creating a video surveillance system

In this section I use three related projects to make a video surveillance system:

Motion (https://motion-project.github.io)
A highly configurable program that monitors video signals from many types of cameras, including network cameras via RTSP, RTMP, and HTTP, as well as Pi Camera Modules and Video4Linux (V4L) USB based webcams.

motionEye (https://github.com/ccrisan/motioneye/wiki)
A web-based front-end for Motion, which lets you configure Motion easily.

motionEyeOS (https://github.com/ccrisan/motioneyeos/wiki)
A Linux distribution that turns a single-board computer such as a Raspberry Pi into a video surveillance system. It essentially combines Motion as a back-end and motionEye as a front-end in a dedicated operating system.

With these projects the Raspberry Pi can be used to create a video surveillance system in two ways:

As an IP camera
You attach a Raspberry Pi Camera Module or another camera to the Raspberry Pi and install motionEyeOS on it to stream video of the camera to your network.

As a camera controller
You install motionEye in a Docker container to collect the output of multiple IP cameras, watch their streams, and invoke scripts when one of the cameras detects action.

Note:
If you want to use a Raspberry Pi as an IP camera, you can install Raspberry Pi OS and motionEye on it together with other software, but this will be less reliable. I recommend using a dedicated Raspberry Pi with motionEyeOS installed on it. This way your camera cannot fail due to other software running on the same system.

You can also just use dedicated IP cameras: these don't have to be based on a Raspberry Pi.

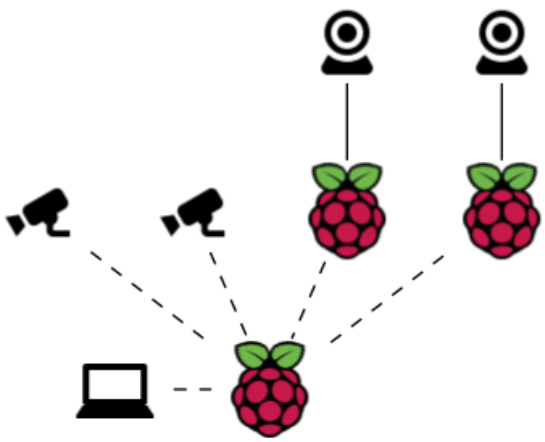

Figure 5.4 With motionEye, one Raspberry Pi can control multiple IP cameras and Raspberry Pis with cameras.

5.5.1 ● Turn your Raspberry Pi into an IP camera

A Raspberry Pi with a camera module and motionEyeOS makes a great IP camera. MotionEyeOS supports all models of the Raspberry Pi, including the original Model B and the Compute Module, as well as a couple of other single-board computers. The project's wiki has a full list of supported devices (https://github.com/ccrisan/motioneyeos/wiki/Supported-Devices), including a link to the newest installation image for each board.

Apart from the board, you need a camera. Motion supports the Raspberry Pi Camera Module, which you connect to the CSI socket on your Pi or webcam that you connect to one of the Pi's USB sockets.[10] Of course, you need a decent power adapter, a microSD card for the operating system, and optionally an Ethernet cable if you don't use Wi-Fi.

Note:
If you want to use this set up as an IP camera, you'll have to use a case that houses both the Pi and camera.

Figure 5.5 A Raspberry Pi with the Pi Camera Module makes an excellent IP camera.

If you use the official Raspberry Pi Camera Module or another camera that uses the CSI port, you need to locate this port first: on a full-size Raspberry Pi, it's next to the HDMI port, white with a black plastic clip on top.

Now gently pull up on the edges of the port's plastic clip until it moves: this opens the port. Then you can insert the camera's ribbon cable vertically. The blue side of the cable

10 I use the Pi Supply Night Vision Camera Module for the Raspberry Pi. It has a 160° fish-eye lens and infrared LEDs. It's supported out-of-the-box in motionEye. Note that right before I finished this book the Raspberry Pi Foundation released its new High Quality Camera. I haven't tested it yet.

should face the Ethernet port. Then push the plastic clip back into its place, firmly locking the cable.

Note:
The Raspberry Pi Zero W has a smaller CSI port. You need an adapter cable to connect a CSI camera to it.

Now that the hardware is ready, on to the software. Download the motionEyeOS image and write it to a microSD card with balenaEtcher (https://www.balena.io/etcher/). See Chapter 2, with the notable difference that you don't install the Raspberry Pi OS now on this Raspberry Pi, but motionEyeOS. After you have written the image to the card, configure Wi-Fi (see Chapter 2 again) if you don't use or have Ethernet on this Pi.

Then put the microSD card in your Pi and boot it. Have a bit of patience: the first time motionEyeOS resizes your data partition and configures various services.

After a while, motionEye is accessible on port 80 of your Raspberry Pi, so you can access it by typing its IP address in your favourite web browser. By default, you can log in with the username **user** or **admin** and an empty password. Log in as **admin**. If the camera is connected correctly, you already see the live camera image.

Warning:
MotionEyeOS doesn't support HTTPS for its web interface. This means that all communication with the Raspberry Pi running motionEyeOS happens unencrypted. However, according to the developer, your username and password are encrypted with a signature mechanism.

First, change some settings. Click on the hamburger menu in the top left and then on **General Settings**. Change the user password and admin password to something secure.

Then change your time zone and set a unique hostname.

If you want to change something to the video output, go to **Video Device**. For instance, if the image is shown upside down, change **Video Rotation** to 180°.

Then go to **Expert Settings**. You can turn off the LED on your camera by disabling **Enable CSI Camera Led**.[11] Set **GPU Memory** to the minimum value of 16 MB if you don't use a CSI camera, or 96 MB if you do. And if you have a CSI camera connected to this Raspberry Pi and you'll use another Raspberry Pi as a camera controller, you can enable **Fast Network Camera** to improve performance. Don't forget to click the orange **Apply** button at the top

11 If you want to turn off the ACT and PWR LEDs of your Raspberry Pi, as well as the Ethernet LEDs, you'll have to change some boot parameters in `/boot/config.txt`. See the appendix at the end of this book for instructions.

to apply your changes.

Note:
The Fast Network Camera option improves performance, but sacrifices advanced features. If you want to use motion detection, picture capturing or movie recording, don't enable Fast Network Camera. If you want to have even better performance than with the Fast Network Camera option, install Raspberry Pi OS on the Pi and then run v4l2rtspserver (https://github.com/mpromonet/v4l2rtspserver) to stream the video with RTSP.

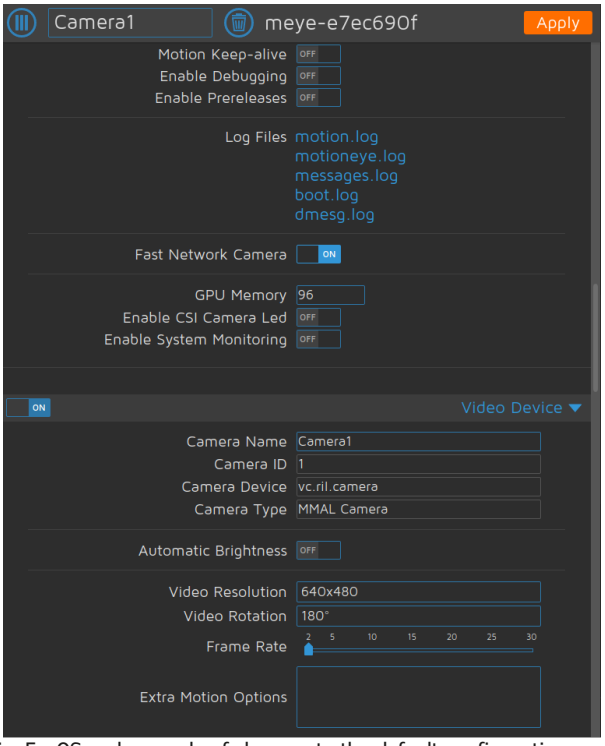

Figure 5.6 With motionEyeOS and a couple of changes to the default configuration, your Raspberry Pi with camera module has become an IP camera.

5.5.2 ● Turn your Raspberry Pi into a camera controller

So now you have turned a Raspberry Pi into an IP camera. If you do this with multiple Raspberry Pis, or if you already have other IP cameras, it would be awkward to have to log into all these devices and look at their video output one by one. Luckily you can also use motionEye(OS) as a camera controller: you can add remote cameras and see them all in one interface.

For the camera controller you can also use motionEyeOS: just repeat the installation and

configuration steps of the previous subsection, but without the camera part; you'll add remote cameras instead. Another option is that you install motionEye as a Docker container on your Raspberry Pi that is working as a home automation gateway. This is what I'll show you next.

Warning:
Just like motionEyeOS, motionEye (which is from the same developer) doesn't support HTTPS for its web interface. This means that all communication with motionEye on your home automation gateway happens unencrypted. Again, according to the developer, your username and password are encrypted with a signature mechanism. To support HTTPS anyway, you could use a reverse proxy in a container on the same machine. See the appendix at the end of this book for a method of doing this.

A Docker Compose file for motionEye could look like this:[12]

```
version: "3.7"

services:
  mosquitto:
    # mosquitto configuration
  motioneye:
    image: ccrisan/motioneye:master-armhf
    container_name: motioneye
    restart: always
    volumes:
      - ./containers/motioneye/etc:/etc/motioneye
      - ./containers/motioneye/lib:/var/lib/motioneye
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "8765:8765"
```

Then create the directory structure for motionEye:

```
mkdir -p /home/pi/containers/motioneye/{etc,lib}
```

Now you can start the Docker container with motionEye:

12 See the previous chapter for the configuration of the mosquitto service that is left out here.

```
docker-compose up -d
```

After this, you can log in to motionEye's web interface on port 8765 of your Raspberry Pi. Again, change the empty user and admin password. Then add a camera by clicking on the message in the main window.

If you have enabled **Fast Network Camera** on the camera of your Raspberry Pi IP camera, choose **Simple MJPEG Camera** as the camera type here and add the URL with port 8081, so http://IP:8081 if IP is your other Pi's IP address. If all goes well, the **Camera** text field should now change to **MJPEG Network Camera**. Click on **OK** to add the camera.

But if you want to have the full features of motionEye, you have left **Fast Network Camera** disabled in the previous subsection, and you can add this remote camera here as a **Remote motionEye Camera**. Enter the URL of the remote motionEyeOS system (http://IP with IP the IP address of that Raspberry Pi) and enter the username **admin** and its password. If all goes well, the **Camera** text field now changes to the name of the camera. Click on **OK** to add it.

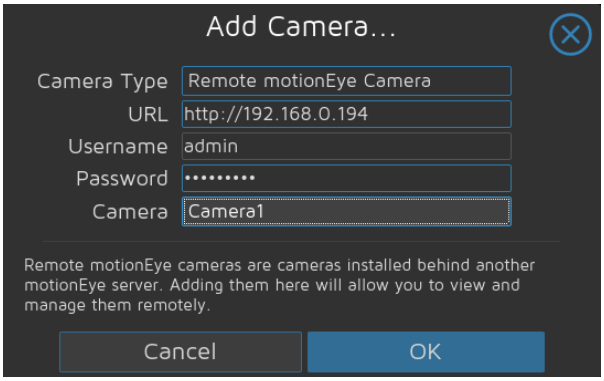

Figure 5.7 You can add remote cameras in motionEye to centrally manage and view all of them.

In the same way, you can add other remote cameras. Just click on the hamburger menu at the top left, then on the text field with the current camera name, and then on **add camera...**. For most IP cameras, the camera type should be **Network Camera**. Those devices stream RTSP/RTMP or MJPEG videos or plain JPEG images. Consult the camera's manual to find out the correct URL. If you have set up a username and password for the camera (as you should), add it here too.

Warning:
If you use a URL beginning with http://, the camera images will be streamed unencrypted over your network. Only do this on a trusted network.

If you have a camera attached to your controller Pi, you can use this camera too. You only have to forward the device on your host to the device in your Docker container. The Docker Compose file should be changed like this:

```
version: "3.7"

services:
  mosquitto:
    # mosquitto configuration
  motioneye:
    image: ccrisan/motioneye:master-armhf
    container_name: motioneye
    restart: always
    volumes:
      - ./containers/motioneye/etc:/etc/motioneye
      - ./containers/motioneye/lib:/var/lib/motioneye
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "8765:8765"
    devices:
      - "/dev/video0:/dev/video0"
```

After a restart of the container with `docker-compose up -d`, the local camera is available for motionEye too.

Remote motionEye Camera is the most versatile camera type. If you have added your remote motionEyeOS camera as this type to your motionEye install on your camera controller, you can not only view the camera image, but also manage the settings remotely, view the pictures and movies, configure motion detection, and so on. If you do this for a couple of cameras attached to Raspberry Pis in your house, you have a very powerful and centrally managed video surveillance system.

5.5.3 ● Viewing your remote cameras

MotionEye's web interface is not the only way to view your cameras: each camera also has video streaming enabled by default. This means that you only have to know the right URL to look at the streaming video in any other program on your network. This gives you a lot of possibilities. For instance, in Chapter 10 you'll see how Home Assistant can integrate and manage your home automation devices. Home Assistant can directly show your camera's video stream and snapshots using the camera integration (https://www.home-assistant.io/integrations/camera/).

You can find these URLs in the **Video Streaming** part of motionEye's settings:

Snapshot URL
It provides a JPEG image with the most recent snapshot of the camera. You can use this URL for instance as the `src` attribute from an `img` element in a HTML page, or in a shell script that automatically downloads this image periodically. The snapshot URL contains a unique code, which you shouldn't share with others: this random code in the URL is the only authentication.

Streaming URL
It provides a MJPEG stream of the camera. You can use this in other programs that support MJPEG streams. If **Authentication Mode** is disabled, everyone with this URL can view the stream. That's why you should enable authentication, for instance by setting it to **Basic**. This will require the user to supply his username and password for motionEye.

Embed URL
It provides a minimal HTML page with the live camera stream. You can put this in an iframe element in another HTML page.

5.5.4 ● Motion detection

By default, motionEye enables motion detection for all cameras, but it doesn't do anything with these motion events. Now, if you want to have a picture taken when the camera detects motion, enable **Still Images** in the settings and change **Capture Mode** from **Manual** to **Motion Triggered** (which takes a series of pictures as the motion is happening) or **Motion Triggered (One Picture)** (which takes one picture when motion is detected).

Warning:
By default, motionEye preserves the pictures forever. This can fill your microSD card very rapidly. It's recommended to change this, for instance to For One Week. You can always check the disk usage in the settings in File Storage.

After you have applied this change, you'll have to start fine-tuning the camera's motion detection settings. Go to **Motion Detection** and first enable **Show Frame Changes**. This will show you a red frame around a region with motion, and you get to see the number of changed pixels in the right top corner of the camera view.

If you want to detect your cat running around in your living room, this number should of course be smaller than if you want to detect a courier ringing your doorbell at your front door. Observe such a situation a couple of times, look when a change is detected (shown by the red frame) and take note of the number of changed pixels.

Figure 5.8 With motion detection enabled in motionEye, you can detect your cats running around at night.

The **Frame Change Threshold** in the motion detection settings is a percentage of your image, so you have to compute this percentage of your number by dividing the number of changed pixels by the total number of pixels in the image (if it's a 640 x 480 camera, that is 307,200 pixels) and multiplying it by 100. For instance, if you see that 1000 pixels is enough to detect your cat's movements and you have a camera with a 1600 x 1200 resolution, the percentage is 1000 / 1600 / 1200 x 100 or around 0,05%. The smallest threshold you can have in motionEye is 0,1%, so set the slider to 0,1% then.

Now you can view the pictures taken in the camera view, by clicking on the **open pictures browser** icon (next to the full screen icon). When you do this the first time, you'll probably see too many pictures (pictures were taken while there was no motion) or too few (not all motion events resulted in a picture being taken). But after you have fine-tuned the frame change threshold, this should only show pictures for movements.

Note:
Don't forget to disable the Show Frame Changes option after you have adjusted the motion detection parameters.

In the pictures browser, you can view, download, and delete each picture individually. By default, they are arranged in a folder by day (this can be changed in the **Image File Name** in the **Still Images** settings). You can download a zip file of them all, and you can even create a timelapse video, which gives you a nice overview of your day.

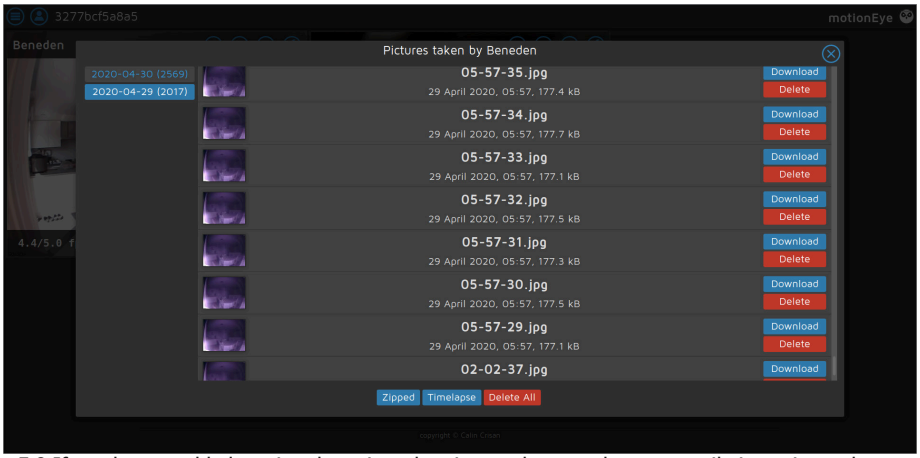

Figure 5.9 If you have enabled motion detection, the pictures browser lets you easily investigate the moments when there was motion.

Sometimes you are only interested in movement in a specific region on the camera image. For instance, you don't want to have a picture created each time a car rides by on the road, but you do want to have a picture when a visitor appears at your front door.

MotionEye has a nice feature to implement this: masks. Go to **Motion Detection** and enable **Mask**. Choose **Editable** for the **Mask Type** and then click **Edit Mask**. Your camera image is now divided by a raster of small blocks. Choose the blocks where you don't want to have movement detected, click on **Save Mask** and then **Apply**.

5.5.5 ● Notifications on motion

So now you have motion detection for your cameras, and you can look at the camera images afterwards, but motionEye can do even more: notify you in real-time when it detects motion. Go to **Motion Notifications** in the settings. Notifications can be done by email, webhook or by using a command. In an email, you can even attach pictures.

If you just want to know whether there's motion or not, enable **Run A Command**. Fortunately, the motionEye Docker container has the `mosquitto_pub` command installed, so you can run the following command to publish an MQTT message to your broker:

```
mosquitto_pub -h mosquitto -u home -P PASSWORD -t "camera/front_door/⏎
motion" -m "ON"
```

This command uses an unencrypted connection to the MQTT broker running on the same Raspberry Pi, which is reachable on the internal Docker network with the hostname `mosquitto`. Just supply the same password as you have configured for Mosquitto in Chapter 4.

In the same way, you can add a command to the **Run An End Command** option:

```
mosquitto_pub -h mosquitto -u home -P PASSWORD -t "camera/front_door/⏎
motion" -m "OFF"
```

If you're now listening to the `camera/+/motion` topic of your MQTT broker, you get `ON` when motion starts and `OFF` when motion ends, for all your cameras:

```
mosquitto_sub -h mosquitto -t "camera/+/motion" -v
```

So now you can let other systems, such as Home Assistant, Node-RED, or your own script, react to motion.

If you want to use this with an encrypted and authenticated MQTT broker running on another device, you have to add your certificates directory as a volume to the Docker container. So change your Docker Compose file to:

```
version: "3.7"

services:
  motioneye:
    image: ccrisan/motioneye:master-armhf
    container_name: motioneye
    restart: always
    volumes:
      - ./containers/motioneye/etc:/etc/motioneye
      - ./containers/motioneye/lib:/var/lib/motioneye
      - /etc/localtime:/etc/localtime:ro
      - ./containers/certificates:/usr/local/share/ca-certificates/:ro
    ports:
      - "8765:8765"
    devices:
      - "/dev/video0:/dev/video0"
```

And recreate the container:

```
docker-compose up -d
```

And then the command to run when motionEye detects motion would become:

```
mosquitto_pub -h HOSTNAME -p 8883 --cafile /usr/local/share/ca-⏎
certificates/rootCA.pem -u home -P PASSWORD -t "camera/front_door/motion" ⏎
-m "ON"
```

Note:
Make sure you use the full hostname, including your network's domain. For instance, instead of `pi-red`, use `pi-red.home`. Otherwise, the Docker container can't find your MQTT broker.

5.6 ● Summary and further exploration

In this chapter, I showed you how to use some TCP/IP protocols to automate things. This is an easy way to start with home automation because you don't need any special transceivers for it. You can wake other devices on your network, execute commands remotely using SSH, collect information using SNMP, control smart devices with a REST API and you can even create a video surveillance system with motionEye.

You can explore many other topics in this way. For instance, instead of Shelly devices, you could buy Sonoff devices (https://sonoff.tech) and flash them with the open-source firmware Tasmota (https://tasmota.github.io/docs/). This lets you control your devices over HTTP, and you can easily connect extra sensors.

Instead of pure HTTP or REST, you could also use a WebSocket API. This provides a full-duplex communication channel over a single TCP connection. In practice, almost all WebSocket implementations work over HTTP(S). You can use the WebSocket protocol for real-time bi-directional data transfer between a client and a web server. In Chapter 12 I give an example of WebSocket communication between the Node-RED automation platform and the Rhasspy voice assistant. You can also create a WebSocket client in Python with the WebSocket-client package (https://pypi.org/project/websocket_client/).

Chapter 6 • Bluetooth

Bluetooth isn't the first protocol you'd think about when you're talking about home automation, but it's quite interesting, especially Bluetooth Low Energy (BLE). Many sensors, fitness trackers and smartwatches send their measurements to other devices using BLE.

The nice thing about BLE sensors is that you don't need anything special on recent models of the Raspberry Pi to read them because all models since the Raspberry Pi 3 have built-in Bluetooth functionality.

One downside about BLE is that it doesn't have a long-range. If you have thick walls in your house, or if you have a sensor in your attic and your home automation gateway is in your basement, chances are that you can't reliably read the sensor values. But there's a cheap solution: just place an extra Raspberry Pi Zero W on the location with bad coverage to be able to read the values.

In this chapter, I illustrate the use of BLE with the Xiaomi Mi Flora plant sensor and the RuuviTag environmental sensor. If you have other BLE devices, you can follow most of this chapter, but you'll have to adapt some examples. My explanation is general enough that this shouldn't be a problem.

6.1 • An introduction to Bluetooth Low Energy

Bluetooth Low Energy has been introduced in 2011 as a subset of Bluetooth 4.0. Sometimes it's called Bluetooth Smart. It's an entirely new protocol compared to the classic Bluetooth. However, the Bluetooth 4.0 specification permits devices to implement both protocols. Moreover, both protocols use the same 2.4 GHz radio frequency, so a dual-mode device can use the same radio antenna for both protocols.

There are two kinds of BLE device:

Peripheral
A low-power, constrained device: for example an environmental sensor or a heart rate sensor

Central
A more powerful device: for example your smartphone or your Raspberry Pi

The Raspberry Pi that you use as a home automation gateway gets the **central** role as a BLE device, while the BLE sensors you want to read have the **peripheral** role.

6.1.1 • Broadcasting data

The fundamental part of the BLE protocol is the Generic Access Profile (GAP), which defines how a Bluetooth device is visible to the outside world. This also defines the data format of the packets that BLE devices can advertise. A GAP payload can contain up to 31 bytes of

data.

It's important to know that BLE devices can send their data without being connected to a specific device. This is called broadcasting: the device (for instance a sensor) sends advertising packets with a custom payload (such as sensor measurements), and every BLE device in the neighborhood can pick up these packets and decode the data. So it's one-to-many data transfer.

BLE devices advertise data continuously with a specific advertising interval. The longer the interval, the less power the device needs, and the longer the battery lasts: the device can sleep most of the time. It just wakes up for sending the advertisement and then goes to sleep again until the next interval starts.

A Raspberry Pi can listen to advertisements and decode the custom data in the payload. This way you can read sensor measurements of a lot of BLE sensors. I'll show you some examples later in this chapter.

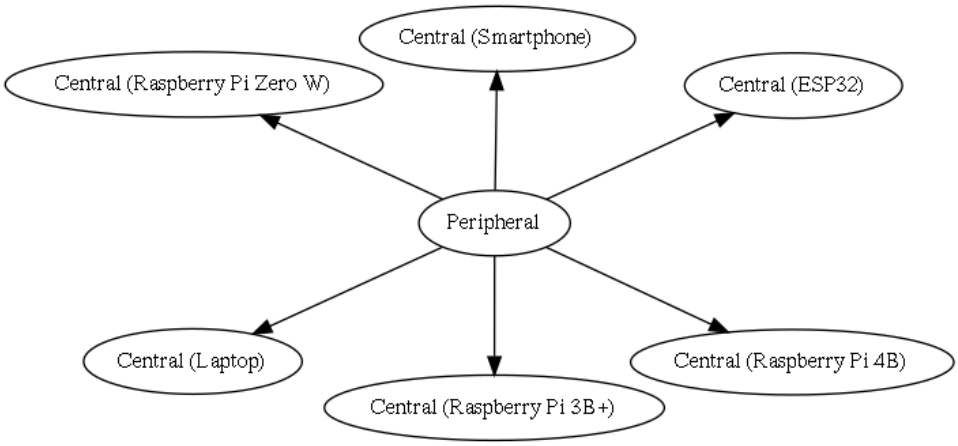

Figure 6.1 A Bluetooth Low Energy peripheral broadcasting data to every device in the vicinity.

6.1.2 ● Connecting to services

Generic Attribute Profile (GATT) defines another way of working with BLE devices: by connecting to them, which is a one-to-one data transfer. This also makes two-way communication possible.

It's important to know that a BLE peripheral can only be connected to one central device at a time and stops advertising as soon as it's connected. That's why other devices can't 'see' a BLE peripheral like a heart rate sensor anymore when it's connected to your smartphone.

A Raspberry Pi can connect to various BLE devices and communicate with them. So if you want multiple devices to communicate with a BLE peripheral using GATT (which isn't

possible), you can let the Raspberry Pi make a GATT connection and act as a central intermediary for other devices. Those then have to communicate to the Raspberry Pi using other means, such as MQTT.

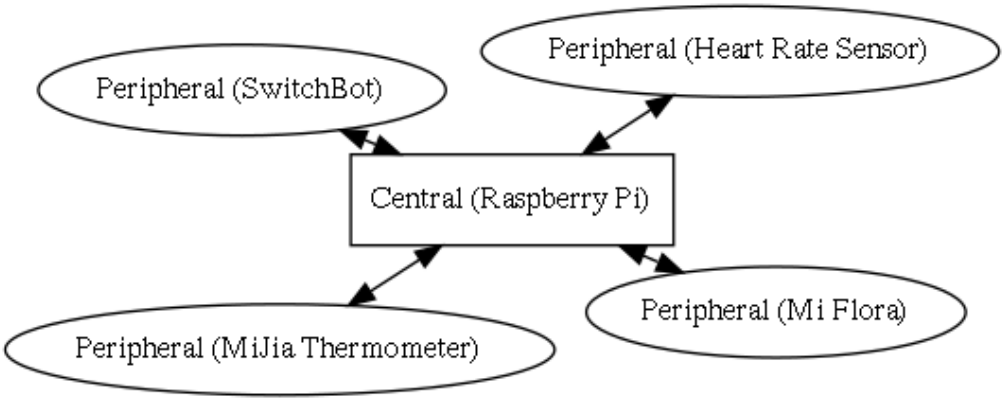

Figure 6.2 A Bluetooth Low Energy central can connect to multiple peripherals in the vicinity and communicate in both directions.

GATT defines three important concepts:

Characteristic
A single value that can be read or written. Each characteristic is distinguished by a UUID (Universally Unique Identifier). A manufacturer of a BLE device can define custom characteristics or use the standard characteristics defined in https://www.bluetooth.com/specifications/gatt/characteristics/. For instance, there are standard characteristics for device name (0x2A00), heart rate measurement (0x2A37), temperature (x2A6E), and so on.

Service
A collection of one or more logically related characteristics. Each service is distinguished by a UUID. A manufacturer of a BLE device can define custom services or use the standard services defined in https://www.bluetooth.com/specifications/gatt/services/. Some standard services are Heart Rate (0x180D), Battery Service (0x180F), and Indoor Positioning (0x1821). A service can have mandatory and optional characteristics. For instance, the Heart Rate service (0x180D) has three characteristics: the mandatory Heart Rate Measurement and the optional Body Sensor Location and Heart Rate Control Point.

Profile
A collection of logically related services. A manufacturer of a BLE device can define custom profiles or use the standard profiles defined in https://www.bluetooth.com/specifications/gatt/. Some examples of standard GATT profiles are Alert Notification Profile, Environmental Sensing Profile, and Find Me Profile. A profile can have mandatory and optional services.

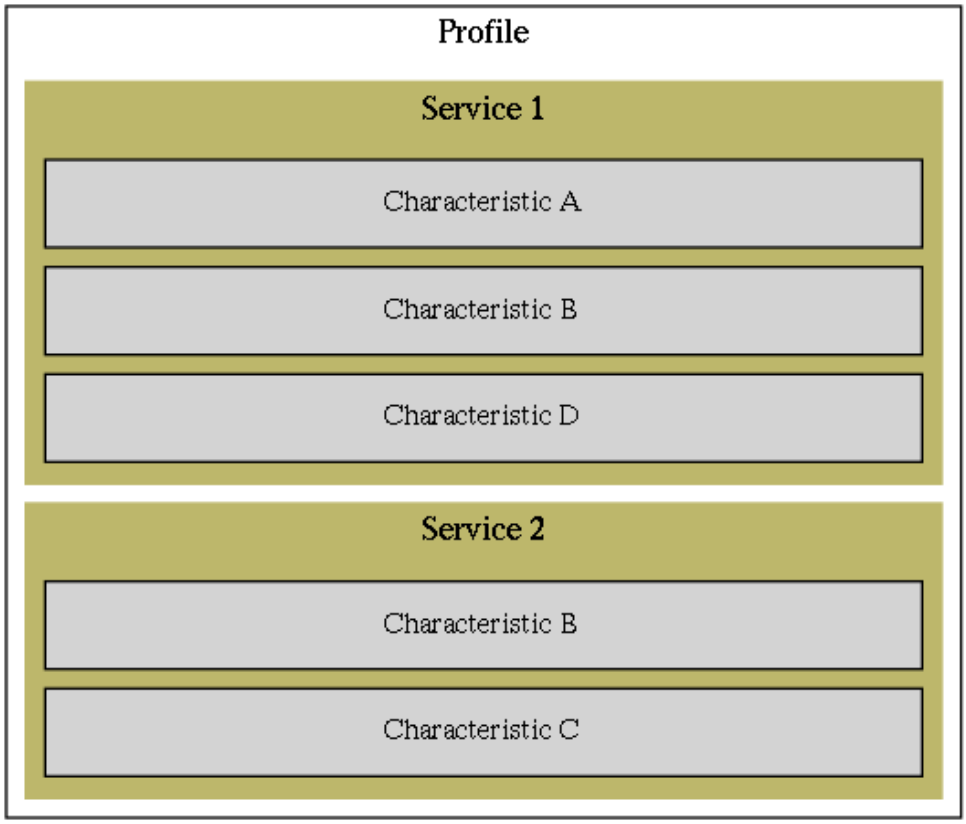

Figure 6.3 A Bluetooth Low Energy profile consists of services, which have characteristics.

6.2 ● Enabling Bluetooth

All models since the Raspberry Pi 3B have an integrated Bluetooth chipset. They all support a Bluetooth version higher than 4.0, and thus support Bluetooth Low Energy:

Raspberry Pi Model	Bluetooth Support
Raspberry Pi 3B	Bluetooth 4.1
Raspberry Pi Zero W(H)	Bluetooth 4.1
Raspberry Pi 3B+	Bluetooth 4.2
Raspberry Pi 3A+	Bluetooth 4.2
Raspberry Pi 4B	Bluetooth 5.0

Table 6.1 Bluetooth support of Raspberry Pi models

If you are using an older model of the Raspberry Pi, you can add Bluetooth functionality with a Bluetooth USB adapter. There's a page with adapters that are known to be working

on https://elinux.org/RPi_USB_Bluetooth_adapters.

Whether you're using the built-in Bluetooth chipset or an external one, check whether it's recognized with:

```
hciconfig -a
```

You should see something like this:

```
pi@raspberrypi:~ $ hciconfig -a
hci0:   Type: Primary  Bus: UART
        BD Address: B8:27:EB:D1:FA:A9  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING
        RX bytes:71630078 acl:0 sco:0 events:3396999 errors:0
        TX bytes:42625149 acl:0 sco:0 commands:2450942 errors:0
        Features: 0xbf 0xfe 0xcf 0xfe 0xdb 0xff 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH SNIFF
        Link mode: SLAVE ACCEPT
        Name: 'raspberrypi'
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Version: 5.0 (0x9)  Revision: 0x122
        LMP Version: 5.0 (0x9)  Subversion: 0x6119
        Manufacturer: Cypress Semiconductor Corporation (305)

pi@raspberrypi:~ $
```

Figure 6.4 Raspberry Pi OS recognizes the Bluetooth chipset.

This shows you the Bluetooth device (**hci0**), the bus (**UART** for a built-in device, **USB** for a USB device), its MAC address, and some statistics. You also see whether the device is **UP**. If for some reason the device is shown as **DOWN**, re-enable it with:

```
sudo hciconfig hci0 up
```

The `hciconfig -a` command is also nice if you're not sure what Bluetooth version your chip has. This is shown in the line beginning with LMP Version.

6.3 ● Investigating Bluetooth Low Energy devices

You can learn a lot about Bluetooth Low Energy devices in the neighborhood with just a couple of tools.

Make sure that you have the relevant packages installed:

```
sudo apt install bluez bluez-hcidump
```

6.3.1 ● Scanning for Bluetooth Low Energy devices

First, you can start scanning for Bluetooth Low Energy devices in the neighborhood:

```
sudo hcitool lescan
```

You get a list of MAC addresses, some of them with a name, others with **(unknown)** as their name. If after a while no new devices appear in the list, interrupt the program with **Ctrl+c**.

You can ask for additional information that a device with a specific MAC address is advertising. For instance:

```
sudo hcitool leinfo C4:7C:8D:67:65:AD
```

This will show something like:

```
pi@raspberrypi:~ $ sudo hcitool lescan
LE Scan ...
F9:DA:D2:0D:62:24 (unknown)
C4:7C:8D:67:65:AD (unknown)
C4:7C:8D:67:65:AD Flower care
C8:03:24:74:7E:0E (unknown)
C5:98:17:63:C3:E3 (unknown)
^Cpi@raspberrypi:~ $ sudo hcitool leinfo C4:7C:8D:67:65:AD
Requesting information ...
        Handle: 64 (0x0040)
        LMP Version: 4.0 (0x6) LMP Subversion: 0x706
        Manufacturer: RivieraWaves S.A.S (96)
        Features: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
pi@raspberrypi:~ $
```

Figure 6.5 Investigate additional information of a Bluetooth device.

One interesting piece of information is the LMP version (Link Manager Protocol). In this case, you're seeing a Bluetooth 4.0 device.

6.3.2 ● Dumping raw Bluetooth broadcast data

One of the things you can do to reverse-engineer Bluetooth broadcast packets is dumping the raw data:

```
sudo hcidump --raw
```

This should result in something like this:

```
pi@raspberrypi:~ $ sudo hcidump --raw
HCI sniffer - Bluetooth packet analyzer ver 5.50
device: hci0 snap_len: 1500 filter: 0xffffffff
> 04 3E 23 02 01 00 00 AD 65 67 8D 7C C4 17 02 01 06 03 02 95
  FE 0F 16 95 FE 31 02 98 00 76 AD 65 67 8D 7C C4 0D AF
> 04 3E 19 02 01 04 00 AD 65 67 8D 7C C4 0D 0C 09 46 6C 6F 77
  65 72 20 63 61 72 65 AF
> 04 3E 21 02 01 03 01 24 62 0D D2 DA F9 15 02 01 06 11 FF 99
  04 03 54 12 28 C7 56 FF FE 00 0D 03 E7 0B 65 AA
> 04 3E 25 02 01 03 01 E3 C3 63 17 98 C5 19 02 01 04 15 FF 99
  04 03 8B 07 02 C7 3F FF B4 FF F0 03 F0 0B 29 00 00 00 00 A5
> 04 3E 21 02 01 03 01 0E 7E 74 24 03 C8 15 02 01 06 11 FF 99
  04 03 4B 15 04 C7 43 FF CB 00 25 04 05 0B 89 A2
> 04 3E 2A 02 01 00 00 DB 53 D6 0B D5 B8 1E 02 01 18 03 19 00
  02 0B 08 4A 42 4C 20 43 68 61 72 67 65 0A FF 57 00 CB 0E BC
  1E 04 00 00 A1
> 04 3E 23 02 01 00 00 AD 65 67 8D 7C C4 17 02 01 06 03 02 95
  FE 0F 16 95 FE 31 02 98 00 76 AD 65 67 8D 7C C4 0D B0
> 04 3E 19 02 01 04 00 AD 65 67 8D 7C C4 0D 0C 09 46 6C 6F 77
  65 72 20 63 61 72 65 AF
< 01 03 0C 00
> 04 0E 04 01 03 0C 00
< 01 03 10 00
> 04 0E 0C 01 03 10 00 BF FE CF FE DB FF 7B 87
< 01 01 10 00
> 04 0E 0C 01 01 10 00 09 22 01 09 31 01 19 61
```

Figure 6.6 The hcidump command lets you receive raw data of Bluetooth broadcast packets.

Without knowing what you have to look at, it's very difficult to get meaningful information out of this stream of raw data.

However, if you look closely you'll discover the MAC addresses of broadcasting peripherals in the data, with the bytes reversed. For instance, a RuuviTag with MAC address C5:98:17:63:C3:E3 will show up in the raw data as:

```
> 04 3E 25 02 01 03 01 E3 C3 63 17 98 C5 19 02 01 04 15 FF 99
  04 03 5C 15 12 C6 24 FF F0 00 10 03 F4 0B 3B 00 00 00 00 A6
```

So if you know which device is broadcasting this data, and if you have another way of interpreting this data (for instance an official app of the device), you can start reverse-engineering the raw data.

Luckily in many cases, you don't have to do this because someone has already done this and has written a library for it. And in the case of the RuuviTag, the Ruuvi company has documented the raw data format extensively on https://github.com/ruuvi/ruuvi-sensor-protocols.

6.3.3 ● Discovering device characteristics

You can also ask for the BLE characteristics of a device with a specific MAC address (in this case a Xiaomi Mi Flora plant sensor):

```
sudo gatttool -b C4:7C:8D:67:65:AD --characteristics
```

As a result, you get a list of numbers that looks daunting at first:

```
pi@raspberrypi:~ $ sudo hcitool lescan
LE Scan ...
C8:03:24:74:7E:0E (unknown)
C4:7C:8D:67:65:AD (unknown)
C4:7C:8D:67:65:AD Flower care
F9:DA:D2:0D:62:24 (unknown)
^Cpi@raspberrypi:~ sudo gatttool -b C4:7C:8D:67:65:AD --characteristics
handle = 0x0002, char properties = 0x02, char value handle = 0x0003, uuid = 00002a00-0000-1000-8000-00805f9b34fb
handle = 0x0004, char properties = 0x02, char value handle = 0x0005, uuid = 00002a01-0000-1000-8000-00805f9b34fb
handle = 0x0006, char properties = 0x0a, char value handle = 0x0007, uuid = 00002a02-0000-1000-8000-00805f9b34fb
handle = 0x0008, char properties = 0x02, char value handle = 0x0009, uuid = 00002a04-0000-1000-8000-00805f9b34fb
handle = 0x000d, char properties = 0x22, char value handle = 0x000e, uuid = 00002a05-0000-1000-8000-00805f9b34fb
handle = 0x0011, char properties = 0x1a, char value handle = 0x0012, uuid = 00000001-0000-1000-8000-00805f9b34fb
handle = 0x0014, char properties = 0x02, char value handle = 0x0015, uuid = 00000002-0000-1000-8000-00805f9b34fb
handle = 0x0016, char properties = 0x12, char value handle = 0x0017, uuid = 00000004-0000-1000-8000-00805f9b34fb
handle = 0x0018, char properties = 0x08, char value handle = 0x0019, uuid = 00000007-0000-1000-8000-00805f9b34fb
handle = 0x001a, char properties = 0x08, char value handle = 0x001b, uuid = 00000010-0000-1000-8000-00805f9b34fb
handle = 0x001c, char properties = 0x0a, char value handle = 0x001d, uuid = 00000013-0000-1000-8000-00805f9b34fb
handle = 0x001e, char properties = 0x02, char value handle = 0x001f, uuid = 00000014-0000-1000-8000-00805f9b34fb
handle = 0x0020, char properties = 0x10, char value handle = 0x0021, uuid = 00001001-0000-1000-8000-00805f9b34fb
handle = 0x0024, char properties = 0x0a, char value handle = 0x0025, uuid = 8082caa8-41a6-4021-91c6-56f9b954cc34
handle = 0x0026, char properties = 0x0a, char value handle = 0x0027, uuid = 724249f0-5ec3-4b5f-8804-42345af08651
handle = 0x0028, char properties = 0x02, char value handle = 0x0029, uuid = 6c53db25-47a1-45fe-a022-7c92fb334fd4
handle = 0x002a, char properties = 0x0a, char value handle = 0x002b, uuid = 9d84b9a3-000c-49d8-9183-855b673fda31
handle = 0x002c, char properties = 0x0e, char value handle = 0x002d, uuid = 457871e8-d516-4ca1-9116-57d0b17b9cb2
handle = 0x002e, char properties = 0x12, char value handle = 0x002f, uuid = 5f78df94-798c-46f5-990a-b3eb6a065c88
handle = 0x0032, char properties = 0x0a, char value handle = 0x0033, uuid = 00001a00-0000-1000-8000-00805f9b34fb
handle = 0x0034, char properties = 0x1a, char value handle = 0x0035, uuid = 00001a01-0000-1000-8000-00805f9b34fb
handle = 0x0037, char properties = 0x02, char value handle = 0x0038, uuid = 00001a02-0000-1000-8000-00805f9b34fb
handle = 0x003b, char properties = 0x02, char value handle = 0x003c, uuid = 00001a11-0000-1000-8000-00805f9b34fb
handle = 0x003d, char properties = 0x1a, char value handle = 0x003e, uuid = 00001a10-0000-1000-8000-00805f9b34fb
handle = 0x0040, char properties = 0x02, char value handle = 0x0041, uuid = 00001a12-0000-1000-8000-00805f9b34fb
pi@raspberrypi:~ $
```

Figure 6.7 The characteristics of a Bluetooth device tell you what it can do

The most important numbers here are the first parts of the UUIDs:

- 2a00
- 2a01
- 2a02
- 2a04
- 2a05
- 0001
- …

Now, look up these numbers in the list of GATT characteristics on the web site of the Bluetooth Special Interest Group (https://www.bluetooth.com/specifications/gatt/characteristics/). Search for each of these numbers in the **Assigned Number** column. The **Name** column describes what kind of value this characteristic corresponds to. For the above list this is:

- Device Name
- Appearance
- Peripheral Privacy Flag
- Peripheral Preferred Connection Parameters
- Service Changed
- …

Only the numbers beginning with **2a** are standard characteristics; the other ones are implemented by the manufacturer in a non-standard way.

6.3.4 ● Reading device characteristics

So according to the Bluetooth specification, the UUID **00002a00-0000-1000-8000-00805f9b34fb** should give us the device name, because **2a00** is the assigned number for the device name. How should you read it? Look in the output of `gatttool` at the value of **char value handle**, in this case, **0x0003**. Then read the value like this:

```
sudo gatttool -b C4:7C:8D:67:65:AD --char-read -a 0x0003
```

In my case, this gives the following output:

```
Characteristic value/descriptor: 46 6c 6f 77 65 72 20 63 61 72 65
```

Those numbers are hexadecimal representations of bytes. Let's convert them to their corresponding ASCII characters:

```
echo "0x46 6c 6f 77 65 72 20 63 61 72 65"|xxd -r
```

This shows **Flower care**, which confirms that I have been investigating the Xiaomi Mi Flora plant sensor.

If you're feeling particularly brave, you can start reverse-engineering the values of all these characteristics, especially the non-standard ones, but for many sensors it's not needed because someone else has already done it. For instance, Xiaomi's Mi Flora sensor is well-supported by a couple of open-source programs. One of these programs is covered in the next section.

6.4 ● Reading BLE sensor values in Python

Reading raw broadcast data or device characteristics using `hcidump` and `gatttool` is fine for experimenting and reverse-engineering, but there's no point in doing this if you just want to read sensor values and someone else has already written useful software to do this.

In this section, I'll show two examples of interesting Python libraries for the RuuviTag and Xiaomi Mi Flora sensors and how to use them in your software.

6.4.1 ● RuuviTag Sensor

The RuuviTag (https://ruuvi.com/ruuvitag-specs/) is an open-source environmental sensor that broadcasts its sensor data using Bluetooth Low Energy. There's a RuuviTag Sensor package for Python, which you can find on https://github.com/ttu/ruuvitag-sensor. It's also available on PyPI.

First, create a virtual environment, activate it, and then install the `ruuvitag_sensor` package in this environment:

```
python3 -m venv venv-ruuvitag-sensor
source venv-ruuvitag-sensor/bin/activate
pip install ruuvitag_sensor
```

After installation, check whether the Python package can find your RuuviTag sensors with:

```
python -m ruuvitag_sensor -f
```

After a few seconds, expect something like this:

Figure 6.8 The ruuvitag_sensor package lets you pick up data broadcasted by RuuviTag sensors.

If this works, you can start using the `ruuvitag_sensor` library in your Python programs.

The RuuviTag not only broadcasts temperature, humidity, and pressure measurements, but it also has an accelerometer. This can be used for some interesting applications. For instance, the Z component of the accelerometer tells you whether the sensor is positioned the right side up, upside down or on its side. If you attach the sensor at the right place on your garage door, this can be used to see whether your garage door is open or closed:

Upside down
Garage door is open

On its side
Garage door is closed

Now you can write a program that sends an MQTT message to your broker when the position of your garage door changes. This could look like this:

```
"""Send an MQTT message when the position of your garage door changes.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
from threading import Thread

import paho.mqtt.client as mqtt
from ruuvitag_sensor.ruuvi import RuuviTagSensor

MQTT_HOST = "HOSTNAME"
MQTT_PORT = 8883
MQTT_CAFILE = "/path/to/rootCA.pem"
MQTT_USERNAME = "home"
MQTT_PASSWORD = "PASSWORD"
MQTT_CLIENT_ID = "RuuviTagGarageDoor"

MQTT_TOPIC = "garagedoor/state"
MAC = "C8:03:24:74:7E:0E"


def handle_data(found_data):
```

```
    """Handle acceleration data from the RuuviTag sensor."""
    global state
    acceleration_z = found_data[1]["acceleration_z"]
    if acceleration_z > 100:
        print("Right side up")
        if state != "error":
            state = "error"
            mqtt_client.publish(MQTT_TOPIC, "error")
    elif acceleration_z < -100:
        print("Upside down")
        if state != "open":
            state = "open"
            mqtt_client.publish(MQTT_TOPIC, "open")
    else:
        print("On its side")
        if state != "closed":
            state = "closed"
            mqtt_client.publish(MQTT_TOPIC, "closed")


def on_connect(client, userdata, flags, rc):
    """Start receiving RuuviTag sensor data after connecting."""
    print("Connected with result code " + str(rc))
    Thread(
      target=RuuviTagSensor.get_datas, args=(handle_data, [MAC]), daemon=True
    ).start()


if __name__ == "__main__":
    # Initialize program state and MQTT connection
    state = "error"
    mqtt_client = mqtt.Client(MQTT_CLIENT_ID)
    mqtt_client.on_connect = on_connect

    # Set up authentication and TLS encryption
    mqtt_client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)
    mqtt_client.tls_set(ca_certs=MQTT_CAFILE)

    # Connect and start event loop
    mqtt_client.connect(MQTT_HOST, MQTT_PORT)
    mqtt_client.loop_forever()
```

If you want to try this in your setup, make sure that you change the constants in the beginning. See Chapter 4 for the first couple of constants regarding the MQTT connection.

Then come the following constants specific for this code:

MQTT_TOPIC
The MQTT topic used to signal the state of the garage door.

MAC
The Bluetooth MAC address of the RuuviTag you have secured to your garage door.

What does the program do? It's quite simple. At the bottom of the file, you create an MQTT client that connects to the MQTT broker and starts a loop until the program is interrupted (for instance if you press **Ctrl+c**). On connecting to the broker, a new thread is started that starts collecting data from the RuuviTag sensor with the specified MAC address. Every time new data are found, the function `handle_data` is called with the found data.

This function extracts the Z component of the acceleration data and derives whether the RuuviTag is right side up (which shouldn't happen in this setup, so it's an error state), upside down (in which case the garage door is open) or on its side (in which case the garage door is closed). In the state differs from the previous state, it's published as an MQTT message on the configured topic.

If you're still in the `ruuvitag_sensor` virtual environment, you just have to install an additional dependency:

```
pip install paho_mqtt
```

Then run the program like this:

```
python ruuvitag_garage_door.py
```

And you can see the state changes of your garage door in your MQTT client on the topic `garagedoor/state`.

Note:
Here's an exercise: how would you adjust the code to be able to send the state of multiple garage doors to the MQTT broker?

6.4.2 ● Miflora

The Xiaomi Mi Flora plant sensor also has a package for Python, which you can find on https://github.com/open-homeautomation/miflora. It's also available on PyPI.

First, create a virtual environment, activate it, and then install the `miflora` package in this environment, together with the `bluepy` library:

```
python3 -m venv venv-miflora
source venv-miflora/bin/activate
pip install miflora bluepy
```

Now make sure that you know your plant sensor's MAC address. Then the following program will show some values of the plant sensor:

```
"""Show some measurements of a Xiaomi Mi Flora plant sensor.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
from btlewrap.bluepy import BluepyBackend
from miflora.miflora_poller import MiFloraPoller

MAC = "C4:7C:8D:67:65:AD"
POLLER = MiFloraPoller(MAC, BluepyBackend)

print("Temperature; " + str(POLLER.parameter_value("temperature")))
print("Moisture: " + str(POLLER.parameter_value("moisture")))
print("Light: " + str(POLLER.parameter_value("light")))
print("Conductivity: " + str(POLLER.parameter_value("conductivity")))
print("Battery: " + str(POLLER.parameter_value("battery")))
```

Run this like this:

```
python3 test_miflora.py
```

After a few seconds, expect something like this:

```
(venv-miflora) pi@raspberrypi:~ $ python test_miflora.py
Temperature; 17.3
Moisture: 11
Light: 254
Conductivity: 28
Battery: 39
```

Figure 6.9 The `miflora` package lets you read data of Xiaomi Mi Flora plant sensors.

If this works, you can start using the miflora library in your Python programs. But it's already integrated into a couple of other projects, one of which I'll show you in the next section.

6.5 ● Relaying Bluetooth sensor values with bt-mqtt-gateway

Using ad hoc Python scripts is interesting if you like tinkering and exploring devices directly, but for a home automation system, you need another approach. One interesting project is bt-mqtt-gateway (https://github.com/zewelor/bt-mqtt-gateway), which provides a gateway that reads Bluetooth sensor values and translates them to MQTT messages.

At the moment, bt-mqtt-gateway supports the following Bluetooth devices:

- EQ3 Bluetooth smart thermostat
- Xiaomi Mi Scale
- Linak Desk
- MySensors devices
- Xiaomi Mi Flora plant sensor (based on the miflora library of the previous section)
- Xiaomi Aqara thermometer
- BLE beacons
- Oral-B connected toothbrush
- Switchbot Bluetooth switch
- Sensirion SmartGadget sensor
- RuuviTag sensor (based on the ruuvitag_sensor library of the previous section)

The architecture is highly extensible: support for other devices can be added by writing custom workers. I have added support for the RuuviTag sensor to the project by copying the worker code for the Sensirion SmartGadget sensor and changing a couple of lines so it would use the ruuvitag_sensor library, which is a testament to the project's extensibility.

Note:
If you are only interested in relaying Mi Flora measurements to MQTT, two excellent projects are miflora-mqtt-gateway (https://github.com/ThomDietrich/miflora-mqtt-daemon) and plantgateway (https://github.com/ChristianKuehnel/plantgateway).

6.5.1 ● Configuring bt-mqtt-gateway

First create a directory for the bt-mqtt-gateway container:

```
mkdir -p /home/pi/containers/bt-mqtt-gateway
```

The `bt-mqtt-gateway` program is configured in a YAML file, `config.yaml`. Create this in

the directory you just created and give it this content:

```
mqtt:
  host: pi-red.home
  port: 8883
  username: home
  password: PASSWORD
  ca_cert: /etc/ssl/certs/rootCA.pem
  topic_prefix: bt-mqtt-gateway
  client_id: bt-mqtt-gateway
  availability_topic: availability

manager:
  command_timeout: 30
  workers:
    miflora:
      args:
        devices:
          aglaonema: C4:7C:8D:67:65:AD
        topic_prefix: miflora
      update_interval: 300
    ruuvitag:
      args:
        devices:
          bedroom: F9:DA:D2:0D:62:24
          livingroom: C8:03:24:74:7E:0E
          terrace: C5:98:17:63:C3:E3
        topic_prefix: ruuvitag
      update_interval: 60
```

At the beginning, under the `mqtt` key, you specify the hostname and port number of the MQTT broker, as well as the username and password. In previous chapters, I showed you how you can just use `mosquitto` as the hostname and 1883 as the port number because you connect to the MQTT broker on Docker's internal network. However, as the bt-mqtt-gateway container has to connect directly to the host network of the Raspberry Pi for its Bluetooth access, you have to use the fully qualified hostname (such as **pi-red.home**) for the hostname, 8883 for the port number and also make sure that you refer to the right CA certificate file in the `ca_cert` line (you'll see the path in the Docker Compose file right away).

Then come the topic prefix, client ID, and an availability topic. The latter is a topic that bt-mqtt-gateway will use to signal when it goes online or offline.

The `manager` key consists of a general timeout value and some workers. In this case, I

defined two workers: `miflora` and `ruuvitag`. For each device, a name is assigned to a MAC address.

Note:
If you want to know the syntax for workers for other devices, have a look at the file `config.yaml.example` on the GitHub repository of the project.

For instance, because I configured `topic_prefix` in the `mqtt` key as `bt-mqtt-gateway` and `topic_prefix` for the ruuvitag worker as `ruuvitag`, and configured a couple of names for the RuuviTag sensors, all measurements of the RuuviTag with MAC address F9:DA:D2:0D:62:24 will be published on the MQTT topic `bt-mqtt-gateway/ruuvitag/bedroom`.

6.5.2 ● Running bt-mqtt-gateway

A Docker Compose file for bt-mqtt-gateway could look like this:[1]

```
version: '3.7'

services:
  mosquitto:
    # mosquitto configuration
  bt-mqtt-gateway:
    image: zewelor/bt-mqtt-gateway
    container_name: bt-mqtt-gateway
    restart: always
    volumes:
      - ./containers/bt-mqtt-gateway/config.yaml:/config.yaml
      - ./containers/certificates:/etc/ssl/certs:ro
    # These capabilities are needed for Bluetooth
    cap_add:
      - NET_ADMIN
      - SYS_ADMIN
    # The Docker host should have working Bluetooth
    network_mode: host
```

This begins all rather straightforward if you have seen the other Docker Compose files

1 If you want to run bt-mqtt-gateway on another device than your main home automation gateway, just drop the mosquitto service, because your MQTT broker is then on your home automation gateway. This could be interesting when your home automation gateway is in a place with bad Bluetooth coverage and you want to read sensor data from Bluetooth devices farther away.

earlier in this book. The second volume of the container refers to the directory with your TLS certificates, so bt-mqtt-gateway can find the CA certificate to verify your MQTT broker's certificate.

Towards the end, there are a couple of new lines. First, you see that the bt-mqtt-gateway container needs some special capabilities `NET_ADMIN` and `SYS_ADMIN`. And second, it needs to run its network in host mode. Otherwise, it won't have access to Bluetooth. Network host mode means that the container has direct access to the network of your Raspberry Pi instead of to an internal network that Docker Compose creates. That's also the reason why you couldn't just use the name of the `mosquitto` container as the hostname in bt-mqtt-gateway's configuration file: this name is only valid as a hostname in the internal network that Docker Compose has created.

Now you only have to run `docker-compose up -d` to start bt-mqtt-gateway. If all goes well, you'll start seeing sensor measurements coming through in your favourite MQTT client.

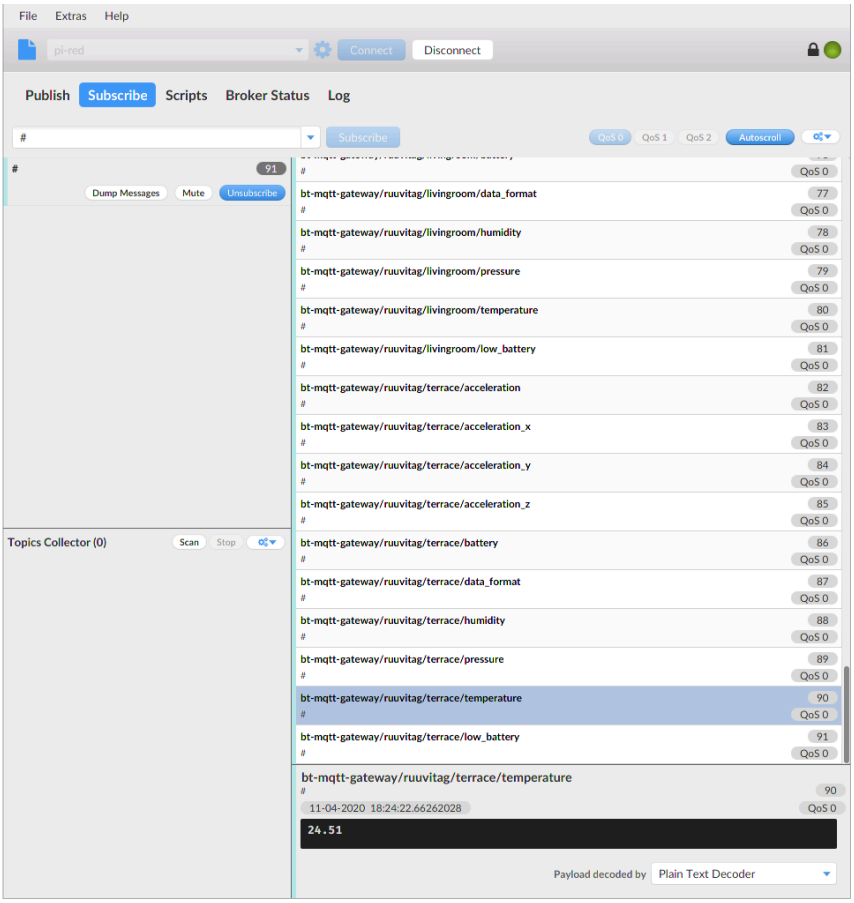

Figure 6.10 With bt-mqtt-gateway running, measurements of your Bluetooth sensors are relayed to MQTT messages.

6.6 • Presence detection with Bluetooth

Bluetooth can not only be used to broadcast sensor measurements but also for presence detection. Indeed, there's a whole class of devices specifically designed to track your location: Bluetooth beacons. A Bluetooth beacon is a BLE device that broadcasts a unique identifier. An app on your smartphone then receives the identifier and knows that you're in the vicinity of that specific beacon. Retail stores use this technology to offer specific information or deals if you're standing at a specific spot.

Now, you could use the same approach at home, with a beacon in every room transmitting its ID, and an app in your phone that receives these broadcasts. But that would mean you'd have to install an app on your phone.

There's another approach: carry a Bluetooth beacon around, and let a device in every room pick up the beacon broadcasts. You can implement this at home by putting a Raspberry Pi in every room (or on every floor) and letting it scan for Bluetooth devices in the vicinity. When you're entering your home with a beacon (or your smartphone) in your pocket, the Raspberry Pi on the first floor detects your presence (if you have Bluetooth enabled on your phone). When you're leaving the house, your phone's Bluetooth signals aren't received anymore by your Raspberry Pi, so it knows you're not home anymore.

When doing this on the first and the second floor, your home automation system can even detect on which floor you are. For each Bluetooth signal that your Raspberry Pi receives, it also gets an RSSI (Received Signal Strength Indicator) value from the device. This RSSI is a rough indication of the distance between the device and the Raspberry Pi. So by comparing the RSSI values of your phone as received by a Raspberry Pi on your first floor and a Raspberry Pi on your second floor, you can estimate whether your phone is on the first or second floor.[2]

6.6.1 • Presence detection with monitor.sh

One project that implements this is monitor.sh (https://github.com/andrewjfreyer/monitor). It detects phones, smartwatches, and laptops, as well as Bluetooth beacons, and reports their presence with MQTT messages.[3]

Monitor.sh works differently for beacons and phones:

Beacons

A Bluetooth beacon periodically broadcasts its ID, which can be considered as a name.

2 In practice the RSSI also varies with the obstacles between the Bluetooth sender and receiver, whether it rains (if you're outside), the orientation of both devices, whether the phone is in your pocket, and each device has unique transmitter and receiver characteristics, so the relation between RSSI and distance should be calibrated.

3 An alternative program for presence detection is Room Assistant (https://github.com/mKeRix/room-assistant). This is not written in Python but in Node.js. There's also a Docker image available. Room Assistant integrates nicely with Home Assistant's `mqtt_room` component (https://www.home-assistant.io/integrations/mqtt_room).

Monitor.sh just listens to these broadcasts and reports on MQTT when a beacon announces its presence.

Phones (smartwatches, laptops)
Contrary to a Bluetooth beacon, a phone doesn't advertise a public ID, but it does respond to a name query. But if you would constantly ask for a specific name, that would interfere with 2.4 GHz Wi-Fi, which is using the same frequencies as Bluetooth. So what monitor.sh does is waiting for some Bluetooth advertisement, and then ask that device whether it has this specific name. If it replies affirmatively, monitor.sh reports its presence on MQTT.

Warning:
Monitor.sh needs exclusive access to the Bluetooth radio to function properly. So you can't run bt-mqtt-gateway and monitor.sh on the same Raspberry Pi. If you need both, I suggest you run monitor.sh on a cheap Raspberry Pi Zero W dedicated to this task.

6.6.2 ● Configuring and running monitor.sh

The monitor.sh project doesn't have an official Docker image yet, but one of the community members has created a Docker image that has the developer's support. You can find it on Docker Hub (https://hub.docker.com/r/mashupmill/presence-monitor).

Monitor.sh can be configured completely with environment variables in the Docker Compose file. This is an example file:[4]

```
version: '3.7'

services:
  mosquitto:
    # mosquitto configuration
  monitor:
    container_name: monitor
    image: mashupmill/presence-monitor:latest
    network_mode: host
    cap_add:
      - NET_ADMIN
      - SYS_ADMIN
    restart: always
    command: ['-b', '-x']
    volumes:
```

4 If you want to run monitor.sh on another device than your main home automation gateway, just drop the mosquitto service, because your MQTT broker is then on your home automation gateway. This could be interesting when your home automation gateway is in a place with bad Bluetooth coverage and you want to monitor Bluetooth devices farther away.

```
    - ./containers/certificates:/etc/ssl/certs:ro
    - /etc/localtime:/etc/localtime:ro
  environment:
    MQTT_ADDRESS: HOSTNAME
    MQTT_PORT: 8883
    MQTT_USER: home
    MQTT_PASSWORD: PASSWORD
    MQTT_CERTIFICATE_PATH: /etc/ssl/certs/rootCA.pem
    MQTT_PUBLISHER_IDENTITY: secondfloor
    KNOWN_BEACON_ADDRESSES: |
      FF:FF:FF:FF:FF:FF Red Tag
      FF:FF:FF:FF:FF:FF Green Tag
      FF:FF:FF:FF:FF:FF White Tag
      FF:FF:FF:FF:FF:FF Black Tag
    KNOWN_STATIC_ADDRESSES: |
      FF:FF:FF:FF:FF:FF Alpha Phone BT
      FF:FF:FF:FF:FF:FF Beta Phone BT
    PREF_ARRIVAL_SCAN_ATTEMPTS: 2
    PREF_DEVICE_TRACKER_REPORT: "true"
```

Just like bt-mqtt-gateway, monitor.sh needs the host network mode and `NET_ADMIN` and `SYS_ADMIN` capabilities.

Make sure that the `MQTT_ADDRESS` environment variable points to the correct hostname, and that you enter the right port, username, password, and path to the root CA certificate. Change these to your situation.

`KNOWN_BEACON_ADDRESSES` lists the Bluetooth MAC addresses of your Bluetooth beacons[5] and `KNOWN_STATIC_ADDRESSES` lists the Bluetooth MAC addresses of your other devices, such as your smartphone, smartwatch or laptop. Don't put a device of one type in the other list, because they behave differently and then monitor.sh wouldn't be able to detect them.

Now if you start monitor.sh with `docker-compose up -d`, you should start seeing MQTT messages on the `monitor/IDENTITY/DEVICE` topic, where `IDENTITY` is the identity you have configured in the `MQTT_PUBLISHER_IDENTITY` environment variable or the hostname of your Raspberry Pi if you haven't done that. `DEVICE` is the name of your device that you have configured in the lists `KNOWN_BEACON_ADDRESSES` and `KNOWN_STATIC_ADDRESSES`.
The message for each device is a JSON object. For instance, when monitor.sh detects the presence of my phone, `mosquitto_sub -t 'monitor/#' -v` shows:

5 This list is optional, because the advertisements of Bluetooth beacons are picked up anyway. But if yours isn't recognized, you can define its MAC address in this list.

```
monitor/raspberrypi/xperia_z5_compact
{"id":"40:B8:37:1C:4F:01","confidence":"100","name":"Xperia Z5
Compact","manufacturer":"Sony Mobile Communications
Inc","type":"KNOWN_MAC","retained":"false","timestamp":"Sun Nov 17 2019
12:00:42 GMT+0100 (CET)","version":"0.2.197"}
```

With this JSON string formatted in a more human-readable way, this becomes:

```
{
  "id": "40:B8:37:1C:4F:01",
  "confidence": "100",
  "name": "Xperia Z5 Compact",
  "manufacturer": "Sony Mobile Communications Inc",
  "type": "KNOWN_MAC",
  "retained": "false",
  "timestamp": "Sun Nov 17 2019 12:00:42 GMT+0100 (CET)",
  "version": "0.2.197"
}
```

You can see here the `id` of the phone (its Bluetooth MAC address), its name, the manufacturer, and the timestamp when it has been detected.

The `confidence` property merits an explanation. This value is the confidence value of the presence of the device. If monitor.sh is 100% sure the device is present, it gives `confidence` the value `"100"`. On the other hand, it monitor.sh is 100% sure the device is absent, it gives confidence the value "0".

If you have set the `PREF_DEVICE_TRACKER_REPORT` environment variable to `"true"`, as I've done in my example Docker Compose file, you also see MQTT topics like `monitor/secondfloor/xperia_z5_compact/device_tracker` with the value `home` or `not_home`. These are useful topics to subscribe to if you want an easy way to know if a device is home or not, and they are used to integrate monitor.sh with Home Assistant (see Chapter 10).

6.6.3 ● Trigger arrival and departure scans in monitor.sh

Monitor.sh has many configuration options to tweak its behavior. If you want to know the details, have a look at the GitHub project, especially the support README file (https://github.com/andrewjfreyer/monitor/tree/master/support/). One interesting functionality is that you can trigger arrival and departure scans. That is, monitor.sh reacts on a message (with blank content, but the content is not important: it's ignored) with these two topics:

`monitor/scan/arrive`
Monitor.sh issues a name request, sequentially, for each device listed in the `KNOWN_STATIC_ADDRESSES` environment variable that is known to be absent.

monitor/scan/depart
Monitor.sh issues a name request, sequentially, for each device listed in the KNOWN_STATIC_ADDRESSES environment variable that is known to be present.

This can be used to make presence detection with monitor.sh more reliable. For instance, you could put a door sensor on your front door, and let your home automation gateway trigger and arrival and departure scan every time your front door is opened.
As I've said in the introduction of this section, you can also run monitor.sh on multiple Raspberry Pis, for instance, one on your first floor and one on your second floor. On each Raspberry Pi, you configure another value for the MQTT_PUBLISHER_IDENTITY environment value.

Then you could let another script look at the RSSI values of the Bluetooth messages picked up by both instances. The instance that has the biggest RSSI is probably the one that is the closest to the device. This way you can deduce on which floor the device (for instance your smartphone) is.

6.7 • Summary and further exploration

In this chapter, I explained the basics of Bluetooth and how you can investigate Bluetooth Low Energy devices. To make this more practical, I illustrated how you read sensor values from the RuuviTag Sensor and the Xiaomi Mi Flora plant sensor with some Python programs. Next, I showed you bt-mqtt-gateway, an interesting project that acts as a gateway that reads Bluetooth sensor values and translates them to MQTT messages. You also learned the basics of Bluetooth presence detection with monitor.sh.

Bluetooth is a vast topic to study, especially in the context of home automation. You can find many cheap Bluetooth gadgets, and it's quite rewarding to try to find out how they work and support them in your home automation setup. This doesn't even have to be that difficult. Just as I was able to add support for the RuuviTag sensor to bt-mqtt-gateway by using an existing library, you could also do this for a Bluetooth device you own.

Presence detection is another fruitful topic to explore. I just covered the basics in this chapter, but if you want to dig deeper you can start experimenting with multiple monitor.sh instances in your house and try to tweak some configuration values to find a setup that works. It would probably be even better to combine Bluetooth presence detection with the detection of your phone on your Wi-Fi network.

With an automation platform such as Node-RED, Home Assistant, or AppDaemon (see Chapter 10) you could combine these values to get better detection rates. Home Assistant is an especially powerful platform to build presence detection on as it has concepts such as a device tracker, a person (to which you can link multiple device trackers) and even Bayesian sensors to estimate the probability of your presence based on the state of multiple other sensors.

Chapter 7 ● 433.92 MHz

It probably sounds weird to name a chapter after a frequency, but in the DIY community (at least in Europe where I'm living), 433.92 MHz will ring a bell. Many cheap wireless devices such as garage door openers, weather sensors, and doorbells are using this frequency. Moreover, the hardware to communicate with these devices is equally cheap.

The disadvantage is that most of these devices use plain unencrypted radio communication, and if they do use some sort of security, it's quite weak and/or some proprietary algorithm that doesn't inspire much confidence. But there are so many available devices and they are so cheap, that you can't ignore them. For security reasons I only use 433.92 MHz temperature sensors: I have one of them in almost every room of my house, and outside too. I wouldn't trust 433.92 MHz devices for critical tasks.

In this chapter, I show you how you read measurements of these wireless temperature sensors and how to relay them to your MQTT broker for further integration in your home automation system.

Note:
In this book, I'm talking about 433.92 MHz, but depending on where you live you have to substitute this by another frequency. For instance, in the Americas, the corresponding frequency is 915 MHz. Just make sure that you buy the correct devices for your country.

7.1 ● 433.92 MHz protocols

Devices that are using the 433.92 MHz frequency operate in the unlicensed industrial, scientific, and medical (ISM) frequency band. But the frequency is one thing, the protocol they're using is another one. There's no standard protocol for this frequency. This is no Z-Wave or Zigbee. However, many protocols of these devices have been reverse engineered, and you can talk to them as long as you have a transceiver for the frequency band around 433.92 MHz and the right software to decode and/or encode the protocol.
Some interesting devices are:

- Temperature, humidity and weather sensors by Alecto, Cresta, La Crosse, and Oregon Scientific;
- Door/window sensors with Hall sensor;
- Switches and dimmers by Energenie, KlikAanKlikUit and LightwaveRF;
- Doorbell chimes by Byron and Chacon.

You can also find many even cheaper devices on AliExpress and Banggood that support the same protocols. And there are even small PCBs such as the STX882 transmitter that you can connect to a microcontroller or an Arduino board to create your own wireless sensor

boards.[1]

In this chapter, I'm focusing on the first types of devices: temperature and humidity sensors. You can find these for less than 10 euros, even for a few euros on AliExpress or Banggood. Their range is quite good: I can read sensors in my whole house, including in my fridge and freezer, and even on my terrace outside.

Figure 7.1 For around €5 you can find a temperature and humidity sensor from DANIU that transmits its values over 433.92 MHz and shows it on a clear display.

7.2 ● Hardware requirements

For your Raspberry Pi to be able to receive 433.92 MHz sensor measurements, you need a receiver and an antenna.

7.2.1 ● Receiver

A popular type of receiver for 433.92 MHz projects is a Realtek RTL2832 based DVB dongle. Yes, you read that right, DVB as in Digital Video Broadcasting. As it turns out, the RTL2832 chip in many of these dongles can do quite more than decoding digital video signals: with the right software, you can create a true software-defined radio (SDR) with it.

So if you have an old DVB dongle lying in your closet, chances are that you can use it to receive signals from your weather sensors. Otherwise, the RTL-SDR (https://www.rtl-

1 Elektor sells a kit with antennas, mounts and extension cable on https://www.elektor.com/rtl-sdr-software-defined-radio-with-dipole-antenna-kit.

sdr.com) is a good choice. You can find variations of this stick for € 25 and in a kit with an antenna and other accessories for € 40 to € 45.[2] You can find something useful even cheaper: I have read about € 7 DVB dongles on AliExpress that are working perfectly for this purpose, but I haven't had any experience with them.

Figure 7.2 The RTL-SDR decodes a lot of wireless signals, including weather sensors transmitting on 433.92 MHz

7.2.2 ● Antenna

The next item you need is a good antenna. There are whole books written about antenna theory, and I'm not going to delve into this vast topic because I'm no antenna specialist. One thing you should know for the choice of your antenna is its length. This depends on the wavelength of the signal. The wavelength equals the speed (in m/s) divided by the frequency (in Hz), and is measured in meters.

Let's do the maths for 433.92 MHz communication. In the air, the speed of the wave is the speed of light. So the wavelength becomes: 299,792,458 m/s / 433,920,000 Hz = 0,69 m. So the full wavelength is 69 cm, the half-wavelength is 34,5 cm, and the quarter-wavelength is 17,25 cm. These are the theoretical optimal lengths for an antenna to receive 433.92 MHz transmissions. In practice, various factors are influencing the antenna's characteristics, including positioning, and there's a rule of thumb to subtract 5% from this theoretical length.

2 Elektor sells a kit with antennas, mounts and extension cable on https://www.elektor.com/rtl-sdr-software-defined-radio-with-dipole-antenna-kit.

Again, this is not an antenna theory book, so I'm not going to talk about the different types of antennas. Moreover, for reading sensor values in your house it doesn't even matter that much what the quality of your antenna is. You could try experimenting with it, but chances are that it just works if you buy a "433 MHz antenna" for a few euros on AliExpress or Banggood. If you don't want to take any chances, use an antenna included in a kit with the RTL-SDR. The official antenna kit has telescopic dipole antennas you can extend from 5 cm to 1 m, which covers the optimal wavelengths for the 433.92 MHz frequency.[3]

Figure 7.3 With the tripod mount, dipole base and telescopic antennas from the RTL-SDR kit, you have all you need to receive measurements from all your 433.92 MHz sensors.

7.3 • Receiving sensor values with rtl_433

On the software side, a popular choice to read 433.92 MHz signals is `rtl_433` (https://github.com/merbanan/rtl_433), which despite its name is a generic data receiver, mainly for the 433.92 MHz, 868 MHz (SRD), 315 MHz, 345 MHz, and 915 MHz ISM bands.

Any Realtek RTL2832 based DVB dongle should work with `rtl_433`, including the official RTL-SDR dongle. I'm using the RTL-SDR dongle with a dipole antenna from the RTL-SDR antenna kit. Just connect the antenna to the RTL-SDR and put the RTL-SDR in a USB port of your Raspberry Pi.

3 There's an extensive explanation of using a dipole antenna kit on the RTL-SDR's web site: https://www.rtl-sdr.com/using-our-new-dipole-antenna-kit/.

Warning:
The RTL-SDR produces quite a lot heat while it's running. Take care of where you position it.

7.3.1 ● Installing rtl_433toMQTT

The `rtl_433` program is actively developed and maintained and has more than 150 protocol decoders for various devices that transmit on 433.92 MHz. Moreover, it can send the received values to an MQTT broker. Luckily, someone created a Docker container with `rtl_433` for this exact purpose (https://github.com/bademux/rtl_433toMQTT).[4]

First, create a directory for the container:

```
mkdir -p /home/pi/containers/rtl433tomqtt
```

Then add the container definition to your `docker-compose.yml` file:

```
version: '3.7'

service:
  mosquitto:
    # mosquitto config
  rtl433tomqtt:
    image: bademux/rtl_433tomqtt:latest
    container_name: rtl433tomqtt
    restart: always
    volumes:
      - ./containers/rtl433tomqtt:/home/user/.config/rtl_433:ro
      - /etc/localtime:/etc/localtime:ro
    devices:
      - /dev/bus/usb:/dev/bus/usb
```

The container needs access to the USB bus to read from the RTL-SDR device. Note also that the directory you created is mounted as a volume. You don't have to create a configuration file yet.

4 Right before this book was finished, the developer announced that he wouldn't actively develop this Docker image (which has more than 100,000 pulls) anymore, so there will be no new features. It remains to be seen if another image becomes popular. But even if you need to find another one, you should only need some minor changes to use it: the main configuration file of `rtl_433` stays the same.

First create a udev rule to give the right permissions to the USB device:

```
sudo nano /etc/udev/rules.d/20.rtl-sdr.rules
```

Enter the following line:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="0bda", ATTRS{idProduct}=="2838",
OWNER="pi", MODE="0660"
```

Save the file with Ctrl+o and exit nano with Ctrl+x. Then unplug the RTL-SDR and reattach it. Now look at the list of attached USB devices:

```
lsusb
```

Search in this list for a line like this:

```
Bus 001 Device 008: ID 0bda:2838 Realtek Semiconductor Corp. RTL2838 DVB-T
```

Note the bus and device number: 001 and 008. Now look at the device files for the bus 001:

```
ls -l /dev/bus/usb/001
```

You should see a line like this:

```
crw-rw---- 1 pi   root 189, 10 May  7 20:22 008
```

This shows that the device is owned by the user `pi`, which has read and write permissions. After this, create the container with:

```
docker-compose up -d
```

After the container has been created, look at its logs:

```
docker logs -f rtl433tomqtt
```

You should see some messages that the `rtl_433` program is trying to find a configuration file in a few places, that is has registered more than 120 decoding protocols and that is has found a receiver device. Then you should see a message "Tuned to 433.920MHz." and if all goes well you should now see sensor values coming in. Have some patience, because many of these sensors only transmit once a minute.

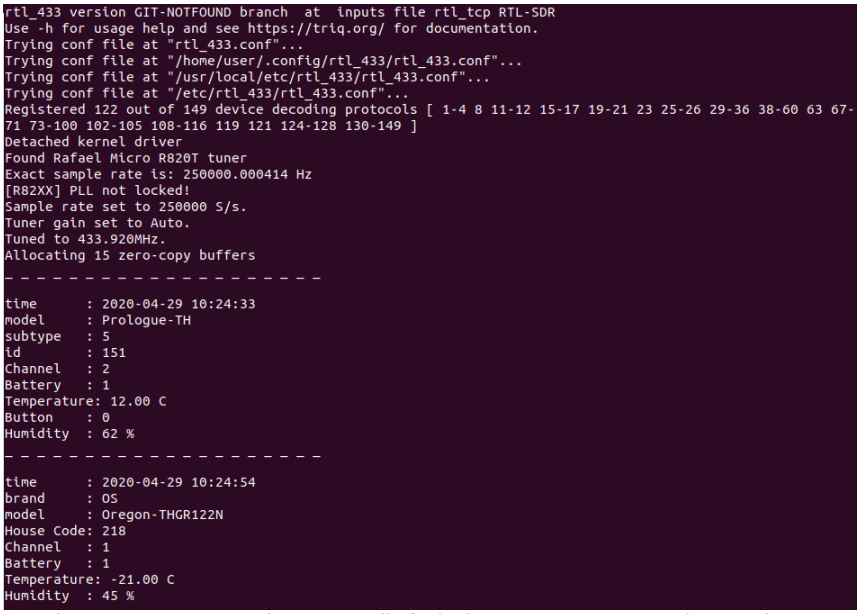

Figure 7.4 The `rtl_433` command automatically finds the RTL-SDR receiver and starts showing received sensor readings.

7.3.2 • Configuring rtl_433

In the beginning of the logs, you saw that `rtl_433` tried to find some configuration files. It didn't find one, so it just used a default configuration, which was fine for testing but didn't use MQTT. Now let's copy an example configuration file to a path where `rtl_433` is searching:

```
docker cp rtl433tomqtt:/usr/local/etc/rtl_433/rtl_433.example.conf /home/
pi/containers/rtl433tomqtt/rtl_433.conf
```

Because I have mapped the directory `containers/rtl433tomqtt` in your pi user's home directory to `/home/user/.config/rtl_433` in the container, you can now edit this

configuration file and restart the container to use this configuration. For instance, you can disable protocols you don't need, or enable protocols that are disabled by default.

The configuration file is heavily commented, which should help you figure out what to change. Moreover, you can find a lot of information in rtl_433's online documentation (https://triq.org/rtl_433/), including steps to add support for a sensor that is not (yet) supported.

After the changes to your configuration file, restart the container:

```
docker restart rtl433tomqtt
```

If all goes well, the logs should show that `rtl_433` stops searching for a configuration file after the second file and finds it:

```
Reading conf from "/home/user/.config/rtl_433/rtl_433.conf".
```

With the example configuration file, `rtl_433` now shows the output of each sensor as a JSON dictionary:

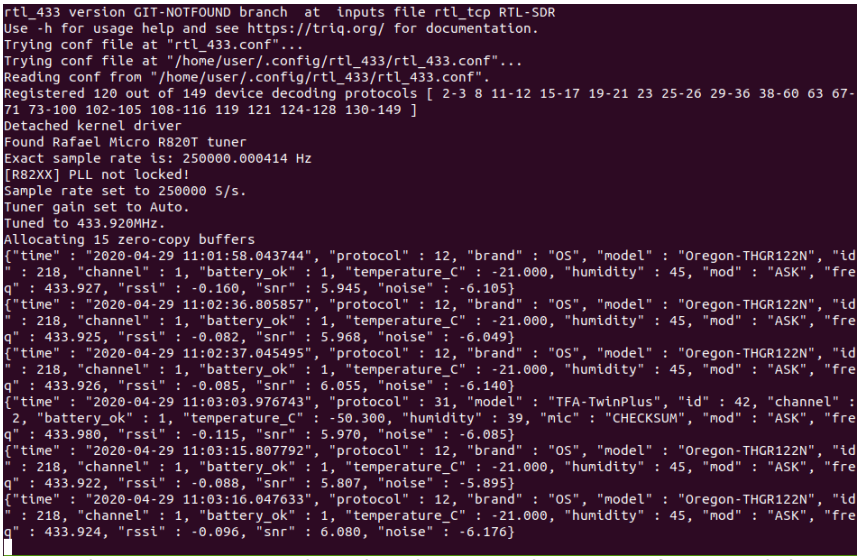

Figure 7.5 The `rtl_433` command can show the sensor values in many formats, including JSON.

This is already a big step forward to integrate your 433.92 MHz sensors into your home automation setup, but the final step is to publish these values to your MQTT broker.

7.4 ● Publishing 433.92 MHz sensor values to MQTT

The `rtl_433` program has already support for sending the sensor values it receives to an MQTT broker. Open the configuration file (`/home/pi/containers/rtl433tomqtt/rtl_433.conf`) and find the line that says:

```
output json
```

You can keep this line here because you can specify multiple outputs, or change it to `output kv` if you prefer the default more human-friendly output. Add the following line to define an MQTT output:

```
output mqtt://mosquitto:1883,user=home,pass=PASSWORD
```

Make sure to enter the correct username and password for your MQTT broker.

Warning:
The `rtl_433` program doesn't support MQTT over TLS. If you're running the rtl_433toMQTT container on the same Raspberry Pi as your mosquitto container, it's no problem that they communicate unencrypted: they're on the same machine anyway. If you have your RTL-SDR receiver on another machine than your MQTT broker (for instance because you have better coverage there), I recommend you run a mosquitto container aside from the rtl_433toMQTT container and configure it as a bridge to your main MQTT broker over an encrypted connection. See the appendix at the end of this book for the details.

After a restart of your container, you should see an MQTT message published under the `rtl_433` main topic for each sensor value. You can find see this with:

```
mosquitto_sub -t 'rtl_433/#' -v
```

For instance, every time the temperature and humidity sensor in my freezer transmits a message, I see:

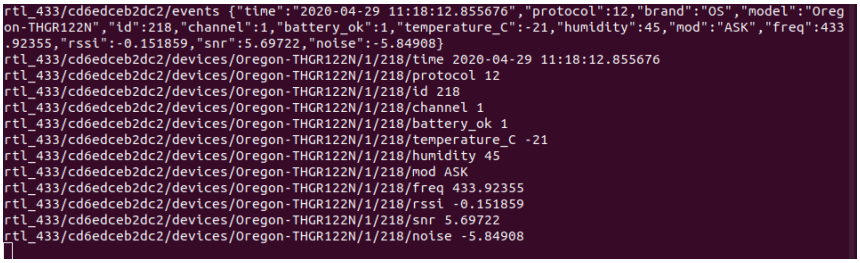

Figure 7.6 The `rtl_433` command can send the received sensor values to your MQTT broker.

Now, you can still tweak a couple of things from the MQTT configuration. Consult the comments in the example configuration file for the details. For instance, this configuration changes the MQTT topics to something shorter:

```
output
mqtt://mosquitto:1883,user=home,pass=PASSWORD,devices=rtl433/[model]/
[channel]/[id]
```

And if you're not interested in the low-level metadata about the radio connection, such as modulation, frequency, RSSI (received signal strength indicator), SNR (signal-to-noise ratio) and noise, just remove or comment out the report_meta level line in the configuration file.

So there you have it: all the measurements of your 433.92 MHz sensors are sent to your MQTT broker, and you can subscribe to their topics in your Python scripts or your other home automation software.

7.5 • Summary and further exploration

Wireless devices in the 433.92 MHz frequency band are very popular in the DIY community and they are cheap and easy to find in a lot of places. Especially interesting are temperature and humidity sensors that you can use to monitor your fridge, freezer, or greenhouse.

All you need to read these sensors is a cheap DVB dongle and an antenna. The `rtl_433` program supports more than 150 protocols of 433.92 MHz devices and lets you relay the received signals as messages to your MQTT broker, so your other home automation software can act upon them.

There are still a lot of interesting topics I barely touched, such as the optimal antenna choice and placement for better coverage. It's also an exciting exercise to try to add support for an unsupported device to `rtl_433`. The project has detailed documentation about how you capture the raw signals and how you should try to reverse engineer the protocol.

You can also try other receivers, for instance, the RFXtrx family of devices, which even

includes a transceiver that lets you control Somfy RTS roller shutters. There's even a Python library, pyRFXtrx (https://github.com/Danielhiversen/pyRFXtrx), to communicate with your 433.92 MHz devices using a RFXtrx transceiver. Home Assistant (see Chapter 10) is using this library for its support of 433.92 MHz devices. However, the RFXtrx transceiver costs a lot more than a RTL-SDR.

At the other end of the price spectrum, if you want to try what you can do with the cheapest possible equipment, the STX882 transmitter and SRX887 receiver are a good place to start.

Chapter 8 • Z-Wave

Although in recent years Zigbee and Wi-Fi have become more popular, Z-Wave is still one of the main wireless home automation protocols. In this chapter, I show you how you can turn your Raspberry Pi into a Z-Wave controller and integrate it with the rest of your home automation system using MQTT.

8.1 • An introduction to Z-Wave

The Z-Wave protocol is a wireless home automation standard originally developed by Zensys, then acquired by Sigma Designs and in 2018 sold to Silicon Labs. Since 2005, all major companies developing Z-Wave devices are grouped in the Z-Wave Alliance (https://z-wavealliance.org). A big advantage of Z-Wave is that all products by companies in the alliance are tested to be interoperable. Each Z-Wave product must pass a stringent conformance test to assure that it meets the Z-Wave standard for complete compliance with all other devices.

Another advantage is that Z-Wave is not using the already crowded 2.4 GHz frequency band, so it avoids interference with the numerous Wi-Fi, Bluetooth, or Zigbee devices in your house.

Z-Wave is specifically designed to provide low-latency transmission of small data packets. That makes it a very suitable protocol for reliable control and sensor purposes.

8.1.1 • The specification

For a long time, Z-Wave has been a closed standard. Manufacturers that wanted to implement the Z-Wave protocol had to sign a non-disclosure agreement before they received access to the Z-Wave Alliance's developer kit. On the software side, OpenZWave became a popular choice for open-source enthusiasts to control their Z-Wave devices.

In recent years the Z-Wave standard has been opened up. In September 2016, certain parts of the standard were made publicly available, which allowed software developers to integrate Z-Wave into devices more easily: Z-Wave's S2 security, Z/IP for transporting Z-Wave signals over IP networks, and the Z-Ware middleware (https://www.silabs.com/products/development-tools/software/z-wave/controller-sdk/z-ware).

At the end of 2019, Silicon Labs and the Z-Wave Alliance announced that the complete Z-Wave Specification would be made publicly available in the second half of 2020, including the ITU.G9959 PHY/MAC radio specification, the application layer, the network layer, and the host-device communication protocol. Before this change, the only company that could make Z-Wave radios was Silicon Labs (and Sigma Designs and Zensys earlier), which meant it had control over the entire ecosystem.[1] After opening up the complete specification, other companies can make Z-Wave radios too. This should drive down prices of Z-Wave products,

1 Although in 2014 Mitsumi received a license from Sigma Designs to manufacture the Z-Wave 500 series chips.

which is welcome because of their rather high price compared to other similar technologies.

8.1.2 ● How does Z-Wave work?

Z-Wave is technologically a mesh network, also called a wireless ad hoc network. Each device wirelessly sends its messages to neighbouring devices, which relay them to the intended receiver. So there's no central router that decides what the optimal path between two devices is. This way Z-Wave devices can communicate even if they're not in each other's range, as long as there's an intermediate device that is in range of the two devices. These routes can even be changed dynamically: if one of the intermediate devices gets out of range, another one relays the messages.

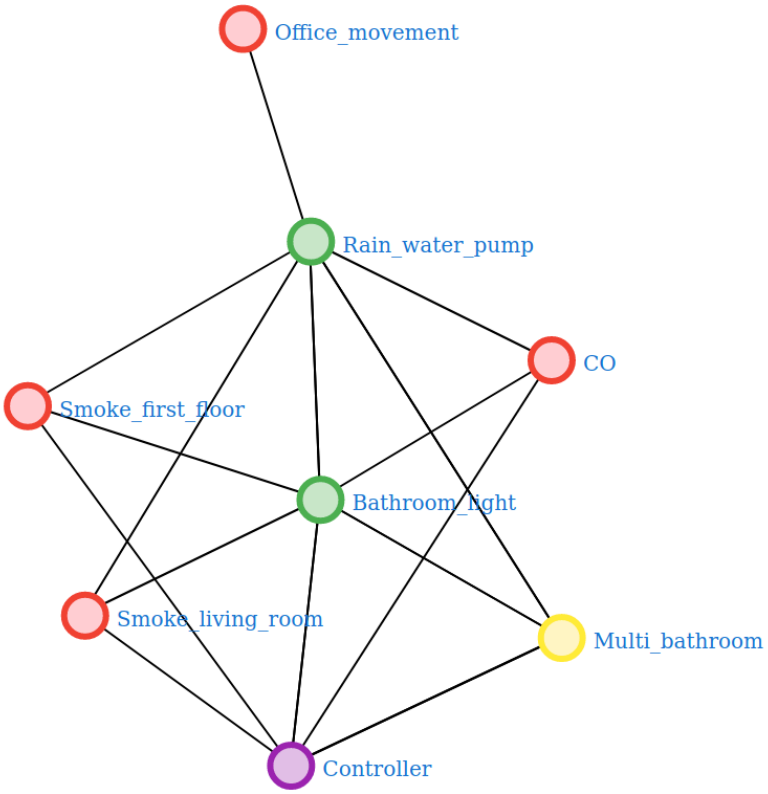

Figure 8.1 Z-Wave is using a mesh network: each device sends or relays messages to neighbouring devices. This is a network graph of a real Z-Wave network, as shown in the web interface of Zwave2Mqtt.

Z-Wave knows two types of devices:

A Controller
This is the central part of the Z-Wave network. It's not like a traditional router (because of the mesh architecture), but it sends commands to the other Z-Wave devices and receives information from them. A controller can be a part of a dedicated gateway device, or it can be a remote control. Each Z-Wave network has one primary controller and optionally one or more secondary controllers.

Nodes
A node is a 'controllable' device, for instance, a sensor or a switch.

A Z-Wave network can consist of up to 232 devices. If you need more devices, you can bridge multiple Z-Wave networks.

Each Z-Wave network is identified by a Network ID, also called Home ID, which is a 32-bits number. For a node to become part of the Z-Wave network, it has to be 'included' by the controller, a procedure that is also called 'pairing'. Usually, this is done by pressing a button on the node and/or on the controller. After this, the primary controller assigns the Network ID to the node, so it becomes part of the network. Nodes with the same Network ID can communicate with each other; nodes with a different Network ID can't. Nodes are removed from the network by unsetting their Network ID.

Each node also has an identifier, the Node ID. This is 8 bits long and is used as an address of the node on the network, hence it needs to be unique in the network.

Z-Wave nodes can be battery-powered (for instance wireless sensors) or using mains electricity (for instance power plugs). Battery-powered nodes are kept in sleep mode as much as possible to consume less energy, and they only wake periodically to perform their task, such as sending a sensor reading. Because nodes can only relay messages when they are not sleeping, only mains-powered nodes are relaying messages on the Z-Wave network. This means that you need to have enough mains-powered Z-Wave nodes that are spatially distributed in your house to get the best range.

Z-Wave radios operate in the unlicensed industrial, scientific, and medical (ISM) frequency band. This means there will be interference with other devices using these frequencies, such as cordless phones. The specific frequencies used by Z-Wave depend on your country's regulations. In Europe, it's 868.42 MHz. In North America, it's 908.42 MHz.

Warning:
If you're buying Z-Wave devices on an international web site, always check whether it's using the correct frequency for your country.

8.2 ● Choosing a Z-Wave transceiver

In this book, you're using a Raspberry Pi as a home automation gateway with a Z-Wave

controller, which is essentially a transceiver (sender and receiver) that sends data packets to your Z-Wave nodes and receives data packets from then.

The most important choice you have to make for a Z-Wave transceiver on your Raspberry Pi is the way to connect it: on the GPIO header of your Raspberry Pi or into a USB socket. Both have their advantages and disadvantages, but the differences are minimal and only relevant in specific situations. Once installed and configured, there's no difference between both options, as the software used to interface with the transceivers is the same.

8.2.1 ● Transceiver on the GPIO header: RaZberry

The most well-known Z-Wave transceiver that sits on top of the Raspberry Pi's GPIO header is the RaZberry (https://z-wave.me/products/razberry/), made by Z-Wave.Me. The RaZberry, currently in its second version, is compatible with all models of the Raspberry Pi. It even works on the original Raspberry Pi.

The RaZberry is a tiny daughterboard that only blocks the first 10 GPIO pins of the Raspberry Pi with its header. It's powered by the Raspberry Pi's 3.3 V pin and typically uses 18 mA of current, with peaks of up to 40 mA while transmitting. It communicates with the Raspberry Pi using UART TTL signals (RX/TX) on pins BCM14 and BCM15 (physical pins 8 and 10 on the header).

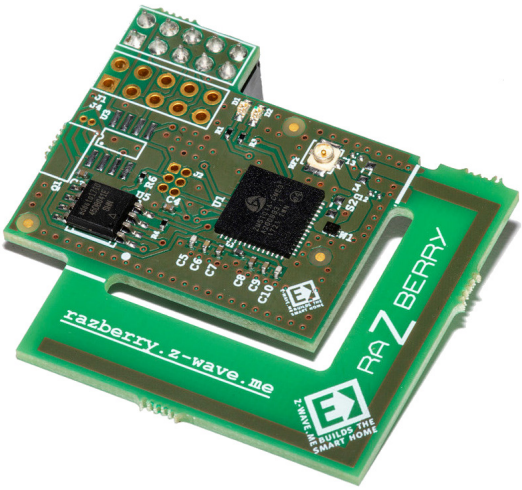

Figure 8.2 RaZberry is a tiny daughterboard with a Z-Wave controller chip that sits on top of the Raspberry Pi's GPIO header.

The heart of the RaZberry is a Sigma Designs ZM5101 Z-Wave transceiver ("5th-generation Z-Wave"). The daughterboard also has two LEDs to indicate the status of the Z-Wave controller chip, an external 32 K SPI flash memory chip, and a PCBA antenna. There's

also an option to solder a U.FL connector on the board if you want to use an external SMA antenna for better coverage.

Warning:
You can't use the Raspberry Pi's on-board Bluetooth chip together with the RaZberry daughter board. Both need the Raspberry Pi's hardware UART to communicate. See the appendix at the end of this book for instructions on how to disable the on-board Bluetooth chip. If you want to use both Bluetooth and Z-Wave on your Raspberry Pi, you have to add a USB Bluetooth adapter or switch your Z-Wave setup to a Z-Wave USB adapter.

8.2.2 ● USB Transceiver

While the RaZberry is specifically designed for the Raspberry Pi[2], a couple of Z-Wave USB transceivers exist that are not designed to fit on the GPIO header, but just slot into a USB socket.

A popular choice is the Z-Stick (https://aeotec.com/z-wave-usb-stick/, currently in its Gen5 version), created by Aeotec, which manufactures many Z-Wave devices.

Figure 8.3 Aeotec's Z-Stick is a popular Z-Wave controller in the footprint of a USB stick.

An advantage of a USB-based Z-Wave transceiver is that you can easily switch it out if you want to use it on another system, for instance, if you want to try it on a second Raspberry Pi or even on a completely different type of computer.

Most of these USB-based Z-Wave transceivers also have a backup battery. So you can take the transceiver off your Raspberry Pi, bring it with you in the vicinity of a Z-Wave node you want to add, and then put it back into your Raspberry Pi.

2 According to Z-Wave.Me, their RaZberry also supports the Orange Pi.

Warning:
Aeotec's Z-Stick seems to violate the USB electrical specification, and as a result, doesn't work on Raspberry Pi 4 if plugged in directly to one of the USB ports. Luckily, a workaround is easy: plug the Z-stick into a USB hub that is plugged into one of the Raspberry Pi's USB ports. The issue doesn't occur on older models of the Raspberry Pi because they use a different USB controller.

8.3 ● OpenZWave and Zwave2Mqtt

I already mentioned OpenZWave (https://github.com/OpenZWave) at the beginning of this chapter: this is a collection of open-source libraries and language wrappers that allow applications to talk to a Z-Wave network using a Z-Wave transceiver.

OpenZWave is a very important piece of software in the world of open-source home automation. The Z-Wave integration of many open-source home automation projects, including Home Assistant (see Chapter 10), is ultimately based on the OpenZWave library. So if you're using a project like Home Assistant that has Z-Wave support directly integrated using OpenZWave, you can use it like this. However, there's another way: with a subproject of OpenZWave, Zwave2Mqtt (https://github.com/OpenZWave/Zwave2Mqtt). This is a fully configurable Z-Wave-to-MQTT gateway and control panel.

Note:
Why would you use Zwave2Mqtt instead of the direct integration with home automation software like Home Assistant, Domoticz or openHAB? If you're this far in this book, I hope that you're already convinced of the approach with MQTT. For Z-Wave in particular this approach has some advantages. For instance, maybe you're still experimenting with Home Assistant and you restart it often, and you want your Z-Wave setup still be functional all this time (Home Assistant takes a lot of time to restart). Or maybe there's bad Z-Wave coverage where your Raspberry Pi running Home Assistant is positioned. Then you can set up a Raspberry Pi running Zwave2Mqtt on a location with better coverage.

Either way, you have to make sure that both your Z-Wave controller and your Z-Wave nodes are supported by OpenZWave:

Controller
Most USB Z-Wave controllers (including various generations of Aeotec's Z-Stick) as well as the Razberry daughterboard are supported. The only real requirement currently is that the Z-Wave controller runs the Static Controller firmware and not the Bridge Firmware. Consult OpenZWave's controller compatibility list (https://github.com/OpenZWave/open-zwave/wiki/Controller-Compatibility-List) for the current status.

Nodes
The OpenZWave project has a device database (http://www.openzwave.com/device-

database/) which lists the supported Z-Wave devices, both nodes, and controllers. Even if one of your nodes is not listed in the database, chances are that it works perfectly.

8.3.1 ● Installing Zwave2Mqtt

Zwave2Mqtt creates a bridge between a Z-Wave network and your local (IP) network using MQTT. It translates Z-Wave messages to MQTT messages on your MQTT broker and vice versa and lets you manage your Z-Wave network using a web interface.

Warning:
Zwave2Mqtt doesn't support HTTPS and at the moment it also doesn't have an option yet to protect the control panel with a password (the developer has promised this feature), so you should only visit its web interface on a trusted network. To add HTTPS and authentication anyway, you could add a Docker container running Nginx (https://www.nginx.com) to your Docker Compose file and configure it to run as a reverse proxy with HTTPS and authentication for your Zwave2Mqtt container. See the appendix at the end of this book for this procedure.

Just like almost all the software in this book, you'll run Zwave2Mqtt in a Docker container. First create a directory for Zwave2Mqtt to store its data in:

```
mkdir -p /home/pi/containers/zwave2mqtt
```

Then edit the `docker-compose.yml` file in your home directory to have the following content: [3]

```
version: '3.7'

services:
  mosquitto:
    # mosquitto configuration
  zwave2mqtt:
    image: robertslando/zwave2mqtt
    container_name: zwave2mqtt
    restart: always
    volumes:
      - ./containers/zwave2mqtt:/usr/src/app/store
    ports:
      - "8091:8091"
```

3 Remember that this is an incomplete Docker Compose file: I don't show the configuration of the mosquitto service anymore.

```
    environment:
      - TZ=Europe/Brussels
    devices:
      - "/dev/ttyUSB0:/dev/ttyUSB0"

      - "8091:8091"
```

Some of these values should be adapted to your situation. For instance, the TZ environment variable should be set to your location's time zone.

But the most important value is the one under the `devices` key. This could be one of `/dev/ttyUSB0`, `/dev/ttyACM0` or `/dev/ttyAMA0`. Change both occurrences to your Z-Wave transceiver's device path, so the container gets access to it. Normally you should see the right path for your setup if you execute the command `dmesg | grep tty`. See the appendix at the end of the book for a better way using the device ID.

Now start the container:

```
docker-compose up -d
```

8.3.2 ● Configuring Zwave2Mqtt

You can now access the web interface of Zwave2Mqtt on your web browser on the URL http://IP:8091. This will show the Control Panel. But first, you have to change some settings. Click on the gear icon on the left You see three categories of settings: **Zwave**, **Mqtt**, and **Gateway**.

Let's go through all of them, beginning with **Zwave**. The most important setting is **Serial Port**. This should be the device path of your Z-Wave transceiver, the same one you have configured in the docker-compose.yml file.

Then there are some switches you can enable or disable: **Logging, Save configuration**, and **Assume awake**. Enable these. Under **Poll interval**, fill in **60000** (60 seconds). The **Config Path** can be left empty.

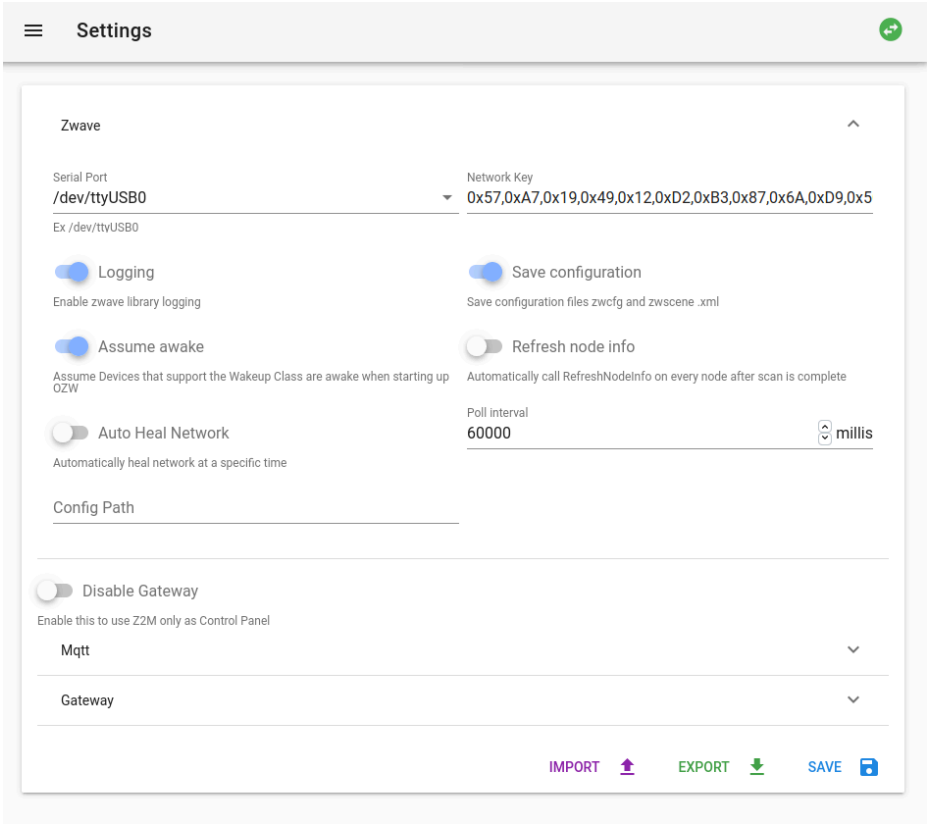

Figure 8.4 Zwave2Mqtt's settings for your Z-Wave transceiver.

If you're using Z-Wave Secure nodes, such as door locks, then you also need to fill in a network key. If you currently have a working Z-Wave network on another controller that you want to migrate to Zwave2Mqtt, you can copy the network key from your current home automation controller and paste it here, under Network Key in the format **0xFD,0x5D,0x45,0xC7,0xFA,0xE9,0xBE,0x07,0x57,0x90,0x84,0xC4,0xA8,0x80,0xBE,0xB3.**

If this is the first time you're setting up your Z-Wave network with a Z-Wave Secure network key, generate a key with the following command on your Raspberry Pi:

```
openssl rand -hex 16 | sed -e 's/\(..\)/0x\1,/g' -e 's/,$//'
```

Don't panic if you don't understand this magical incantation. It's enough to know that it generates a pseudo-random number and converts it to the right format so you can use it as a Z-Wave network key.

Then copy the output of this command and paste it under **Network Key** in the Z-Wave settings of ZWave2Mqtt.

Warning:
Always make sure that you have a backup of your Z-Wave network key. If you lose it, you can't connect to your Z-Wave Secure nodes. The only way to use these devices again with another network key is to give them a factory reset.

Now that these low-level Z-Wave settings have been configured, move over to the **Mqtt** part. If you run Zwave2Mqtt on the same Raspberry Pi as the MQTT broker, fill in a name (without spaces), enter mosquitto as the host URL, and 1883 as the port.

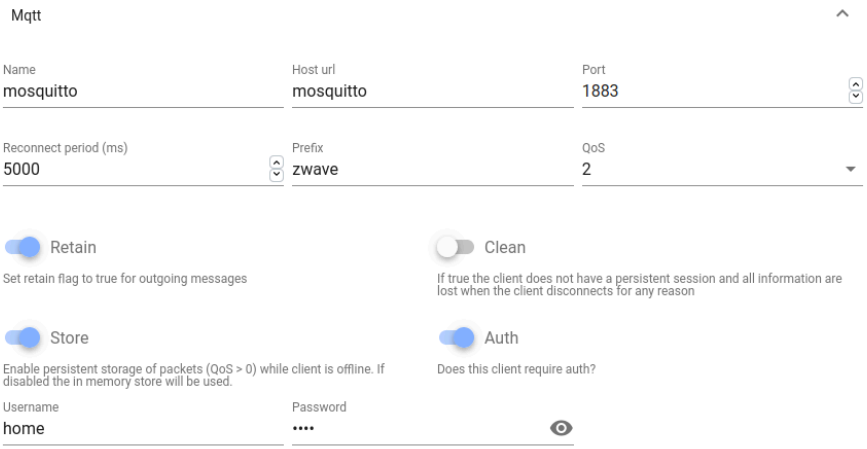

Figure 8.5 Zwave2Mqtt's settings for the connection with your internal MQTT broker.

If you run Zwave2Mqtt on another device than your MQTT broker, fill in a name (without spaces), fully qualified hostname, and port number (8883 for encrypted MQTT, 1883 for unencrypted MQTT). If you use MQTT over TLS, add the protocol to the hostname, for instance, **mqtts://pi-red.home**. After this, a couple of new fields for TLS certificates appear. Click on the **CA.PEM** field and choose the **rootCA.pem** file that you have copied from your Raspberry Pi to your computer.

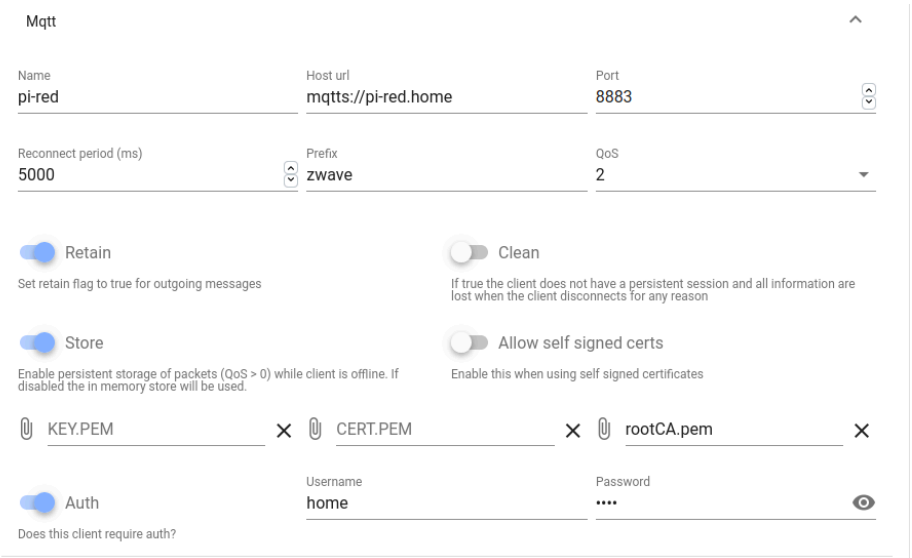

Figure 8.6 If you run Zwave2Mqtt on another device than your MQTT broker, you need to configure TLS for the connection.

After these first connection settings, enable **Auth** and enter the username and password for your MQTT broker.

For **Reconnect period (ms)** you can use for example **5000** (5 seconds). Choose a **Prefix** that will be used for all MQTT topics that ZWave2Mqtt generates, for instance, **zwave**. For **QoS** you can choose **2**. Enable **Retain** and **Store**.

In the last section, **Gateway**, you configure Zwave2Mqtt's translation between the Z-Wave and MQTT messages. The most important choice you have to make here is the **Type** field. If you set it to **ValueID topics**, Zwave2Mqtt automatically chooses MQTT topics based on the properties of your Z-Wave nodes.

The advantage of the **ValueID topics** is that you don't have to manually configure all your Z-Wave nodes to get your system working. But these auto-generated topics are not very human-readable by default, so it's recommended to also enable **Use nodes name instead of numeric nodeIDs**.

In **Payload type** you can choose which data the MQTT messages contain. If you want to see as much data as possible, choose **Entire Z-Wave value Object**. If you want to have Zwave2Mqtt seamlessly integrated with Home Assistant (see Chapter 10), switch on **Hass Discovery**.

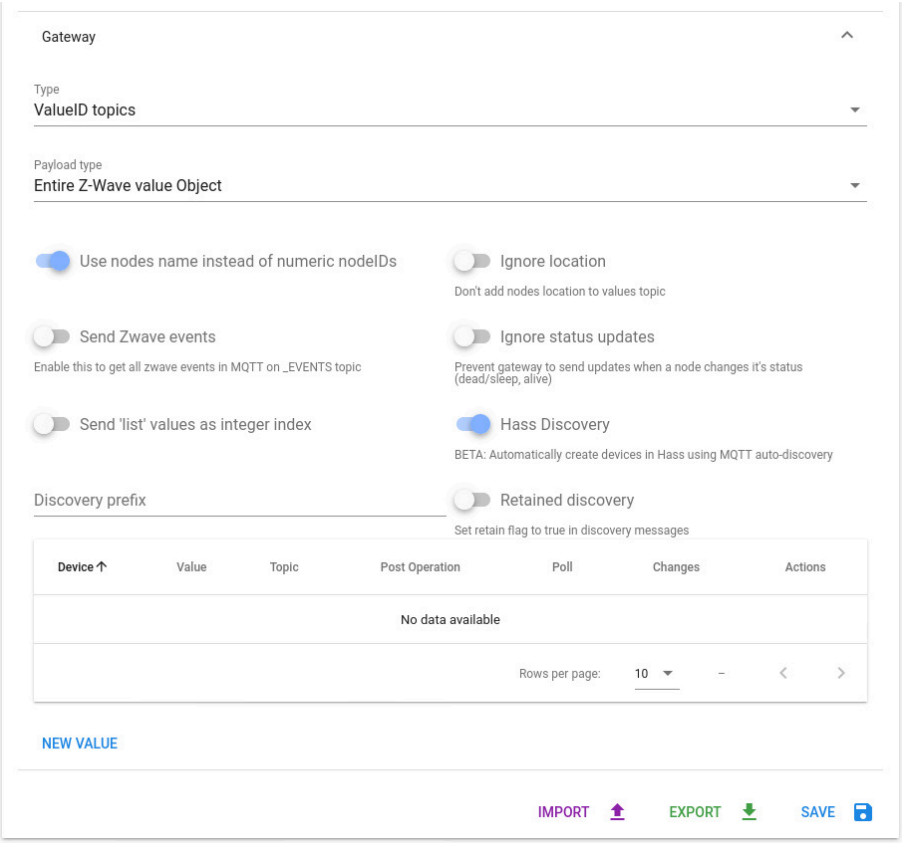

Figure 8.7 Zwave2Mqtt's gateway settings define how the translation between Z-Wave and MQTT messages happens.

Finally, click on the **Save** button at the bottom to save all your settings.

8.3.3 ● Using the Zwave2Mqtt Control Panel

After you have configured the settings, click on the icon with the four squares at the top left. This opens the Control Panel. If all goes well and if you already paired Z-Wave nodes to your controller in the past, you'll see (some of) your Z-Wave devices in the list, including your controller.

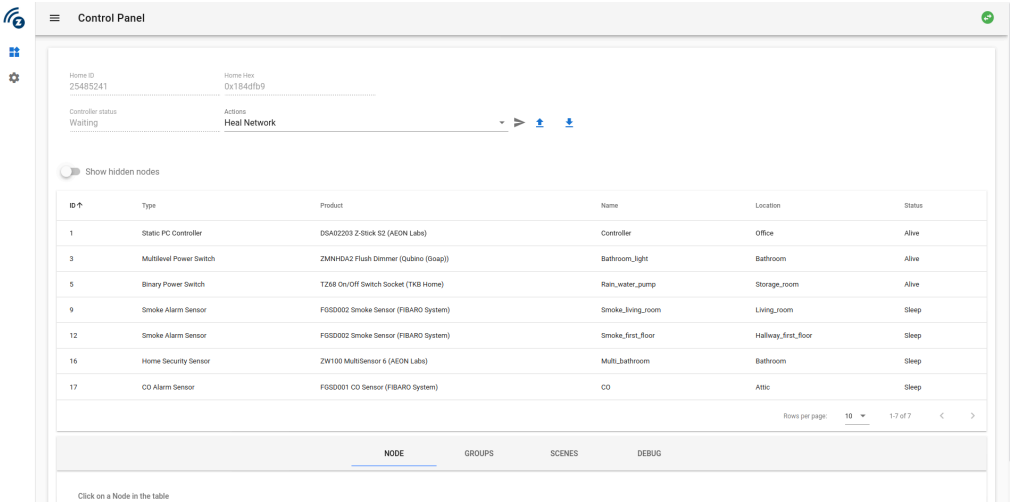

Figure 8.8 Zwave2Mqtt's control panel lets you control specific Z-Wave nodes.

Adding new nodes is easy. Click on the **Actions** menu, choose **Add Node (inclusion)**, and click on the arrow on the right. Most Z-Wave nodes now require you to push a button once or three times quickly. Consult your device's user manual for this pairing procedure. If the node has been added correctly to the network, the **Controller status** changes to **Completed**. It can take a while before the node appears in the list.

If you click on a node in the list, the tab **Node** at the bottom lists some information about your device and lets you take some actions. Most of the relevant information is in the entities under **Values**:

User
Here you can see the values of a sensor or change the state of a switch.

Configuration
Here you can change the device's parameters, such as a sensor's sensitivity, thresholds, intervals, or the behavior of status LEDs.

System
Here you'll find system information such as the supported Z-Wave version and you can configure low-level settings such as a timeout value or a power level.

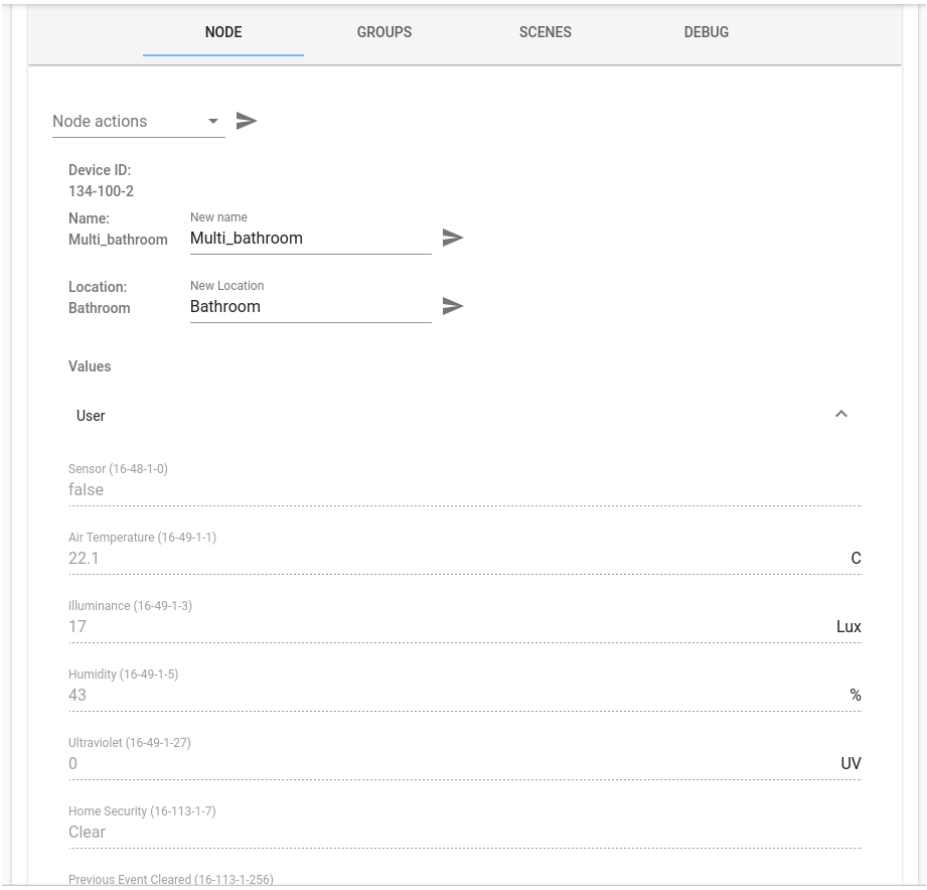

Figure 8.9 Zwave2Mqtt shows information about each specific Z-Wave node.

You can also assign a name (don't use ", +, *, and spaces in the name) and location to your node. Just enter the fields and click on the arrow at the right. If you have enabled **Use nodes name instead of numeric nodeIDs** in the gateway configuration, this human-readable name is used to identify the device in the MQTT messages and the control panel, for instance in the table with nodes.

You can also execute various actions on a node. Just choose the right action in the **Node actions** drop-down menu and click on the arrow at the right.

If you encounter some problems with a Z-Wave node, open the **Debug** tab. Click on **Start** to start receiving debug messages. If you have found a solution to your problem, don't forget to click on **Stop** so the debug logs aren't needlessly filling your Raspberry Pi's microSD card.

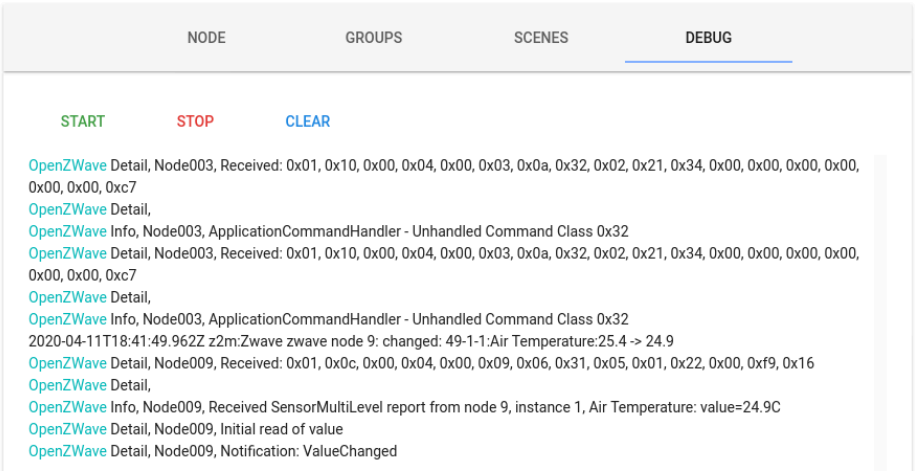

Figure 8.10 You can debug a specific Z-Wave device in the Zwave2Mqtt control panel.

Another way to debug problems is by using the logs of the Docker container, for instance with:

```
docker logs -f zwave2mqtt
```

The output you get would look something like this:

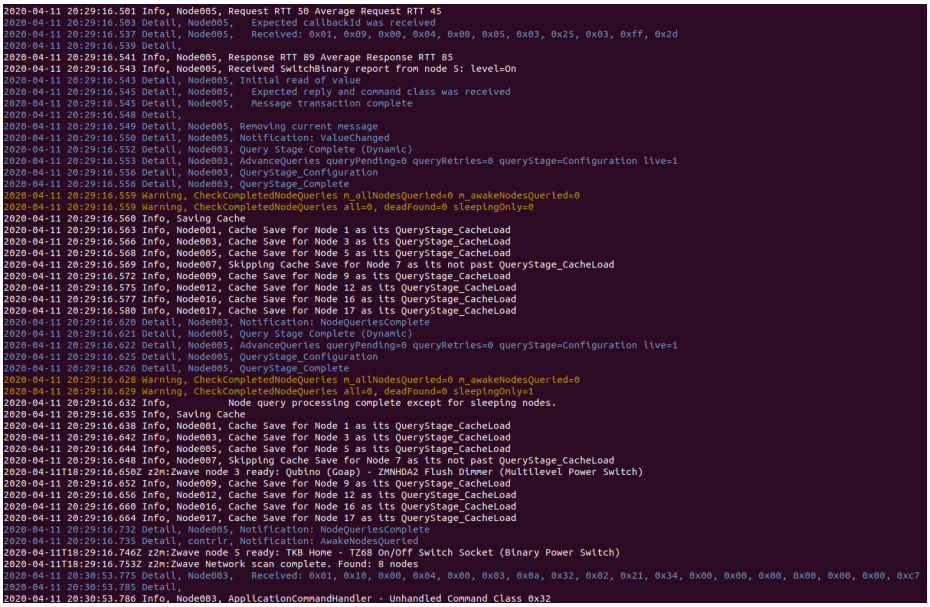

Figure 8.11 Zwave2Mqtt shows a lot of logs, which is helpful to debug problems with your Z-Wave network.

Note:
If you have enabled **Hass Discovery** in the gateway configuration, you'll see at the bottom (under **Home Assistant - Devices**) the virtual devices that Zwave2Mqtt creates for this Z-Wave device in Home Assistant. One Z-Wave device can become multiple devices in Home Assistant. For instance, a so-called multisensor can have a temperature sensor, movement sensor, light sensor, and so on. See Chapter 10 about using MQTT (and hence Zwave2Mqtt) in Home Assistant.

8.4 ● Using your Z-Wave devices with MQTT

Now that you have Zwave2Mqtt running and have your devices configured, it's time to start using the system with MQTT.

Let's first see what MQTT messages Zwave2Mqtt emits, for instance when a Z-Wave sensor sends a value. Have a look with the `mosquitto_sub` command:

```
mosquitto_sub -t 'zwave/#' -v
```

Use the same prefix as the one you configured in the MQTT settings of Zwave2Mqtt, in this example **zwave**.

The output you get would look something like this:

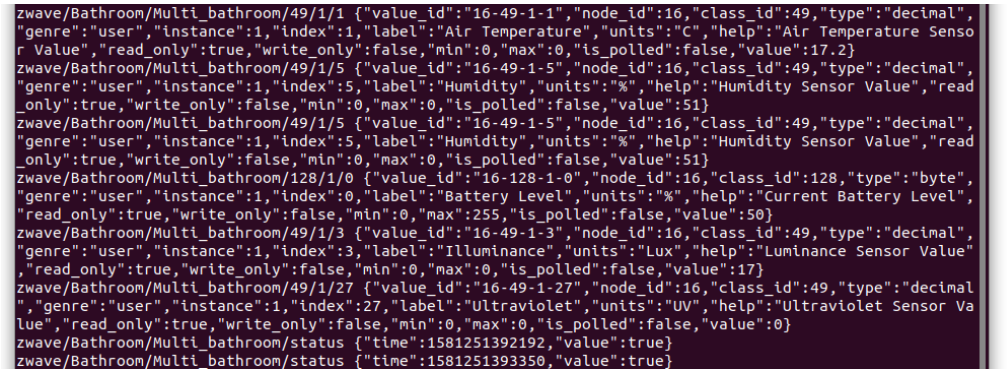

Figure 8.12 Zwave2Mqtt translates Z-Wave messages to MQTT messages, and vice versa.

This looks quite intimidating, but that's partly because it's the output of a multisensor, which packages many sensors in one device. Let's fully dissect the first MQTT message. The topic is `zwave/Bathroom/Multi_bathroom/49/1/1`.

Looking at the hierarchical components of the topic, you see:

zwave
The prefix you configured in Zwave2Mqtt's settings.

Bathroom
The location you assigned to the device in the node's settings.

Multi_bathroom
The name you assigned to the device in the node's settings.

49
The node's class ID.

1
The node's instance.

1
The node's index.

You can find the latter three numbers in the node's user values in Zwave2Mqtt's control panel. With this sensor, it shows the value **Air Temperature (16-49-1-1)**. Apart from the first number (**16**, which is the node ID), these numbers come back in the MQTT topic. So if you want to create a program that listens to the temperature of this sensor, you only have to subscribe to the `zwave/Bathroom/Multi_bathroom/49/1/1` topic.

8.4.1 ● Reading sensor values

Now that you know what the hierarchical components of the MQTT topics mean, let's take a look at the message itself. This is a long JSON string, so I have created a more human-readable representation of it with the `jq` command, which shows each value on its own line:[4]

```
{
  "value_id": "16-49-1-1",
  "node_id": 16,
  "class_id": 49,
  "type": "decimal",
  "genre": "user",
  "instance": 1,
  "index": 1,
  "label": "Air Temperature",
  "units": "C",
  "help": "Air Temperature Sensor Value",
  "read_only": true,
```

4 You can also have these JSON messages formatted like this in MQTT.fx by choosing **JSON Pretty Format Decoder** on the bottom right.

```
  "write_only": false,
  "min": 0,
  "max": 0,
  "is_polled": false,
  "value": 17.2
}
```

Here you see again the node ID, class ID, instance, and index, as well as the full value ID, which is **16-49-1-1** in this case.

But the more interesting information is in some of the other values. For instance, you can find the value of the sensor in **value**, the unit of measurement in **units**, a short description in label and a longer description in **help**, and you can even see if it's a read-only value or if you can change it by publishing an MQTT message to the topic yourself.

In this case, reading the MQTT message tells you that it's a sensor reading of a temperature sensor, and the temperature is 17.2 degrees Celsius.

If you want to use these values in your programs, you can easily subscribe to the relevant MQTT topics with the Paho MQTT Python library, parse the JSON content of the messages and react to the values and/or use the label and units to show these values in a human-readable way.

8.4.2 ● Controlling switches

Now how do you set a value, for instance, if you want to turn on a switch? First, you have to know the topic of the specific switch. For instance, `zwave/Storage_room/Rain_water_pump/37/1/0`. Make sure that it's not a read-only value, and find out what type of value it expects, such as a boolean value (true or false), an integer or a decimal value.

Setting the value then is simple: just write the value (for instance `true` to turn on a switch) to the topic with `/set` after it. On the command line, this would look like this:

```
mosquitto_pub -t 'zwave/Storage_room/Rain_water_pump/37/1/0/set' -m 'true'
```

After this, the Z-Wave switch turns on, because Zwave2Mqtt translates the MQTT message you sent to the corresponding Z-Wave command to the right node.

Note:
You will also receive a new MQTT message on zwave/Storage_room/Rain_water_pump/37/1/0 with the new value.

You can use the same approach to dim lights, setting heating setpoint temperatures, and so on. Doing this in Python with Paho MQTT is straightforward. See Chapter 4.

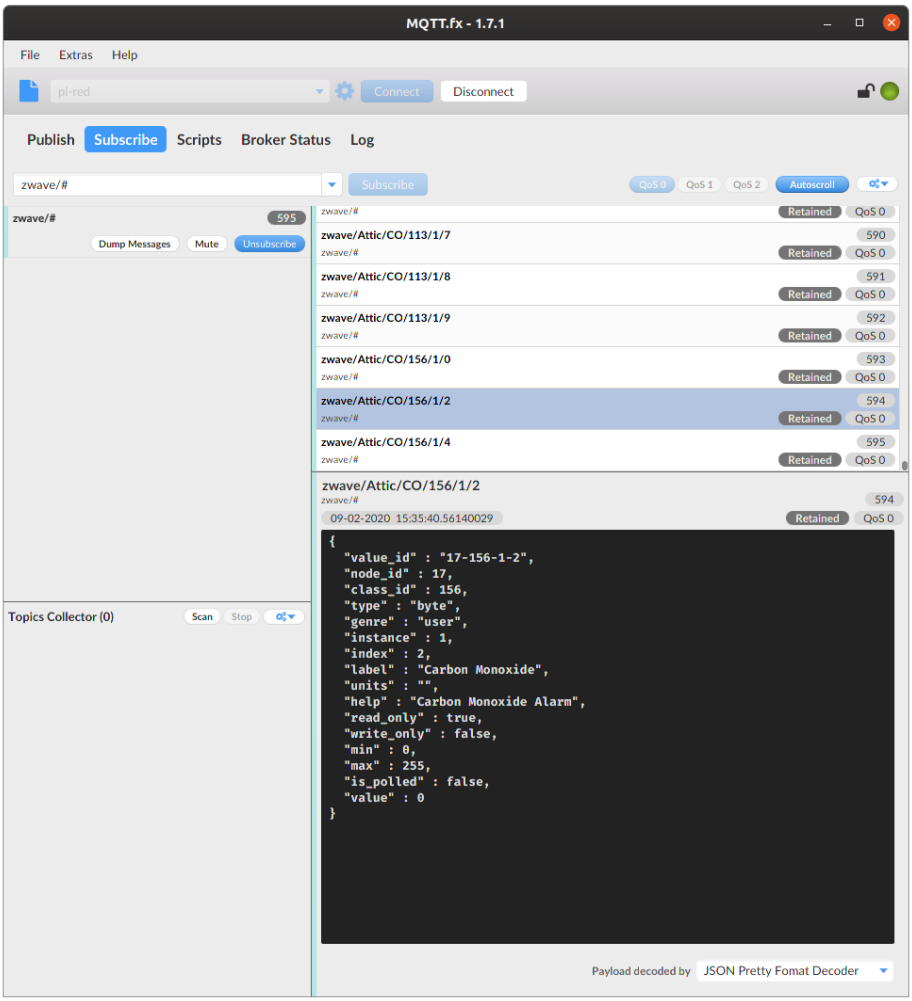

Figure 8.13 A graphical MQTT client like MQTT.fx is very helpful to try to understand the various MQTT messages that Zwave2Mqtt sends.

8.5 • Summary and further exploration

Z-Wave is an interesting home automation protocol supported by many high-quality devices. With a Z-Wave transceiver connected to your Raspberry Pi and the Zwave2Mqtt software, you turn your Raspberry Pi into a Z-Wave controller.

With Zwave2Mqtt's web-based control panel, you can configure all your Z-Wave devices and manage your Z-Wave network. After this, you can read sensor values by subscribing

to MQTT topics and control switches and lights by publishing to MQTT topics. This makes it easy to integrate your Z-Wave devices with the rest of your home automation system.
If one of your Z-Wave devices isn't supported by OpenZWave, that shouldn't be a big concern. See the wiki page "Adding Devices" (https://github.com/OpenZWave/open-zwave/wiki/Adding-Devices) on how to add support for a new device. It boils down to adding a configuration file with the information you find in the user manual or technical specification that comes with the product you bought. By adding your device configuration to OpenZWave, you can use it in Zwave2Mqtt and you help other owners to do the same.

Chapter 9 • Zigbee

Zigbee has become a quite popular home automation protocol in recent years. Many people are using it, even if they don't know it. For instance, the popular Philips Hue and IKEA TRÅDFRI lamps are based on Zigbee technology. Compared to Z-Wave (see the previous chapter), Zigbee devices are quite affordable too.

Many people use Zigbee with a gateway of the manufacturer of their Zigbee devices, such as the Philips Hue Bridge. However, that's yet another device in your home, and why do you need another gateway if your Raspberry Pi can do this? This is why I show you how you can create your own Zigbee gateway with your Raspberry Pi in this chapter.

9.1 • An introduction to Zigbee

Zigbee is a standard for short-range and low-power wireless communication with a low data rate between devices. It's popular in home automation, but also industrial applications such as energy monitoring. The Zigbee protocol can use a couple of frequency bands, but most Zigbee devices for home automation operate in the 2.4 GHz band.

Warning:
Zigbee shares its 2.4 GHz frequency band with Bluetooth and Wi-Fi. This may result in interference and thus low performance and in the worst case, unreliable networks. You can find more about Zigbee and Wi-Fi coexistence on https://www.metageek.com/training/resources/zigbee-wifi-coexistence.html.

The standard is maintained and published by a group of companies that have formed the Zigbee Alliance (https://zigbeealliance.org). It's not an open standard: only members have access to the complete Zigbee specifications, and they have to pay annual membership fees. This is a challenge for free and open-source software developers.

9.1.1 • The specification

Zigbee builds on the lower-level physical and media access control layers defined in the IEEE 802.15.4 standard for low-rate wireless personal area networks (LR-WPANs). The Zigbee specification adds layers on top of this for device discovery, network joining, security, and so on.

Version 1.0 of the Zigbee specification has been ratified in 2004. Since then, the specification has been updated with so-called application profiles. The Zigbee Alliance has defined a profile for various application domains, including home automation, smart energy management, building automation, and toys. Not only are there public profiles. A manufacturer can also define a profile specific for its own purpose.

9.1.2 ● How does Zigbee work?

Just as Z-Wave, Zigbee is technologically a mesh network, also called a wireless ad hoc network. Each device wirelessly sends its messages to neighbouring devices, which relay them to the intended receiver. So there's no central router that decides what the optimal path between two devices is. This way Zigbee devices can communicate even if they're not in each other's range, as long as there's an intermediate device that is in range of the two devices. These routes can even be changed dynamically: if one of the intermediate devices gets out of range, another one relays the messages.

Zigbee knows three types of devices:

A coordinator
This is the central part of the Zigbee network: this device starts the network, stores information about the network, and can bridge the Zigbee network to other types of networks. Every Zigbee network has precisely one coordinator.

A router
This device routes traffic between different devices. Apart from this, it can also have another function, such as being a lamp or a sensor. A coordinator has all router capabilities, but you can add extra routers to a Zigbee network to improve coverage.

End devices
An end device has a function, such as a sensor or switch, but it is not able to relay data from other devices. Each end device has one parent, the coordinator, or a router, and it communicates with the network via this parent.

Battery-powered devices are typically end devices: they remain asleep as long as possible and only wake up temporarily from time to time to execute their function, such as sending a sensor measurement. Afterwards they return to sleep.

Mains-powered devices, such as Philips Hue or IKEA TRÅDFRI lamps, are typically routers. Because they don't have to consume as less power as possible, they can stay awake continuously and relay data they receive from other devices.

9.2 ● Creating a Zigbee transceiver

In this book, you'll use a Raspberry Pi as a home automation gateway. To add Zigbee support, and essentially to turn your Raspberry Pi into a Zigbee coordinator, you need a Zigbee transceiver. There are a couple of options (see https://www.zigbee2mqtt.io/information/supported_adapters.html), but in this chapter, I'll only talk about the CC2531 USB stick, because this is the option with the best support in Zigbee2mqtt (https://www.zigbee2mqtt.io), the software that you'll use.

The CC2531 is not an out-of-the-box Zigbee coordinator, so you need some extra components and you have to follow a couple of steps before you have a usable Zigbee

system. First, you need to purchase the following hardware:

- CC2531 USB sniffer
- Downloader cable for the CC2531
- Four (or five) female-to-female jumper wires
- Male-to-female USB extension cable

These are all available on AliExpress for a couple of dollars per component.

9.2.1 ● Connect the downloader cable

The downloader PCB is a small PCB with 10 full-size header pins on one side and another smaller header with 10 pins on the other side. It comes with a fine ribbon cable: attach it to the small header pins. Position the cable in such a way on the header that the cable points away from the PCB.

Attach the other side of this ribbon cable to the CC2531, again with the cable pointing away from the PCB.

Warning:
The small header pins on the downloader PCB and the CC2531 are quite vulnerable. Attach the ribbon cable slowly on both ends, so you don't bend or break any pins.

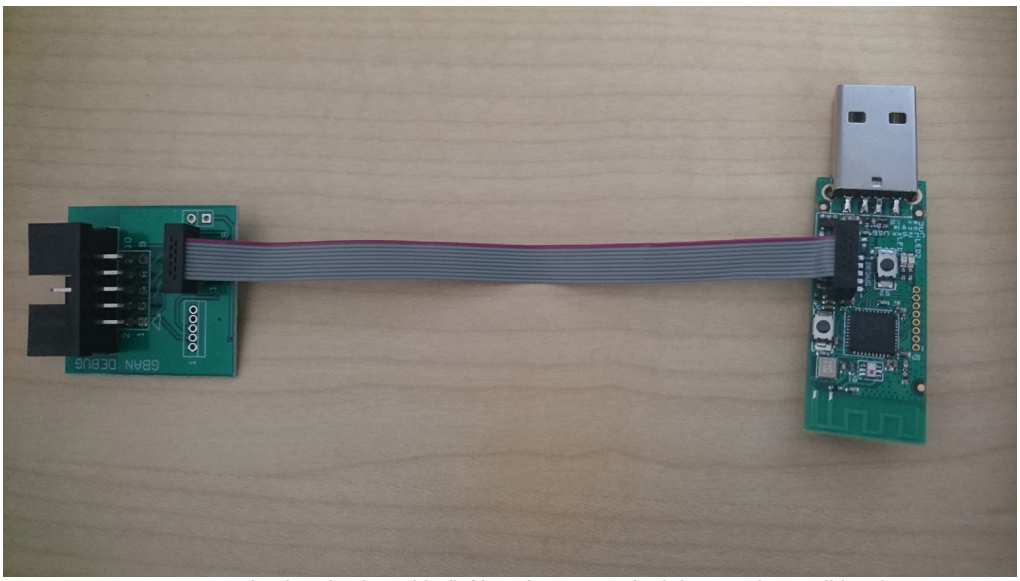

Figure 9.1 Connect the downloader cable (left) to the CC2531 (right) using the small header pins.

Now connect the full-size header pins of the debug port of the downloader cable to your

Raspberry Pi with jumper wires, like this:

Downloader PCB	Raspberry Pi GPIO
pin 1 (GND)	pin 39 (GND)
pin 7 (reset)	pin 35 (GPIO24, BCM19)
pin 3 (DC)	pin 36 (GPIO27, BCM16)
pin 4 (DD)	pin 38 (GPIO28, BCM20)

Table 9.1 Pin connections between downloader cable and Raspberry Pi

For the downloader PCB, the pin numbers are printed on the PCB behind the header. For the Raspberry Pi, consult the GPIO pinout on https://pinout.xyz.

After you have connected the jumper wires, connect the CC2531 to a USB port of the Raspberry Pi to power it.

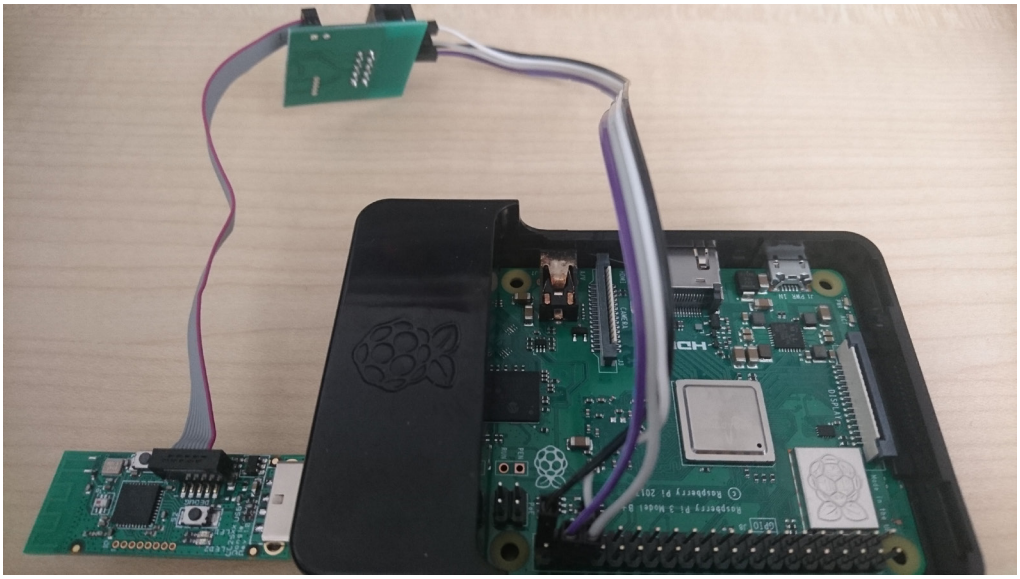

Figure 9.2 Connect the CC2531 and the downloader cable to the Raspberry Pi to flash new firmware to the USB sniffer.

Note:
You can add a fifth connection: between pin 2 (Target Voltage Sense) on the downloader PCB and pin 1 or pin 17 (3V3) on the Raspberry Pi. This powers the CC2531 using your Raspberry Pi's GPIO pins instead of the USB connector. You can even do the whole procedure without a downloader cable, but then you need to bend the small debug pins outwards because they are too close together to connect the jumper wires.

9.2.2 ● Install the flashing software

On the software side, first, install the wiringPi library on your Raspberry Pi:

```
sudo apt install wiringpi
```

Then download the software to flash the CC2531 from GitHub:

```
git clone https://github.com/jmichault/flash_cc2531.git
cd flash_cc2531
```

Double-check whether you have connected the downloader cable and CC2531 correctly. Then test whether the flashing software recognizes your CC2531:[1]

```
./cc_chipid
```

It should return:

```
ID = b524.
```

If you see `ID = 0000.` or `ID = ffff.` in the output, something is wrong and you should probably check your wiring. If it still doesn't work, consult the project's GitHub repository (https://github.com/jmichault/flash_cc2531).

9.2.3 ● Flash the firmware

Koen Kanters, the developer of Zigbee2mqtt, has put some custom firmware in the project Z-Stack-firmware (https://github.com/Koenkk/Z-Stack-firmware). You can find the firmware for a coordinator in the directory `coordinator`.

In this book, I use the `Z-Stack_Home_1.2` firmware, which implements the Zigbee Home Automation 1.2 standard. With a CC2531 this is enough to support 20 direct children with one coordinator. Download the correct zip file on your Raspberry Pi. At the moment when I'm writing this chapter, this is the following:

1 This uses the precompiled version of the tool. You can compile it yourself with the `make` command.

```
wget https://github.com/Koenkk/Z-Stack-firmware/raw/master/coordinator/Z-⏎
Stack_Home_1.2/bin/default/CC2531_DEFAULT_20190608.zip
```

Then unzip the file:

```
unzip CC2531_DEFAULT_20190608.zip
```

This results in two files: `CC2531ZNP-Prod.hex` and `CC2531ZNP-Prod.bin`.

With both the downloader cable and the CC2531 USB sniffer connected to your Raspberry Pi, you can erase and flash the firmware on the CC2531:

```
./cc_erase
./cc_write CC2531ZNP-Prod.hex
```

This takes a couple of minutes.

```
pi@raspberrypi:~/flash_cc2531 $ unzip CC2531_DEFAULT_20190608.zip
Archive:  CC2531_DEFAULT_20190608.zip
  inflating: CC2531ZNP-Prod.hex
  inflating: CC2531ZNP-Prod.bin
pi@raspberrypi:~/flash_cc2531 $ ./cc_write CC2531ZNP-Prod.hex
  ID = b524.
  reading line 15490.
  file loaded (15497 lines read).
writing page 128/128.
verifying page 128/128.
 flash OK.
```

Figure 9.3 I just flashed the Z-Stack coordinator firmware for Zigbee Home Automation 1.2 on the CC2531 USB sniffer.

Afterwards you can remove the downloader cable and the CC2531. Disconnect the downloader cable too from the CC2531.

9.3 ● Zigbee2mqtt and Zigbee2MqttAssistant

I already mentioned Zigbee2mqtt: as its name implies, this project lets you interact with Zigbee devices using MQTT, which fits directly into the main approach I use in this book.

First, you have to make sure that both your Zigbee coordinator as your other devices are supported by Zigbee2mqtt. The CC2531 is the main supported Zigbee coordinator in Zigbee2mqtt, and for the other devices, you should consult the list of supported devices on

https://www.zigbee2mqtt.io/information/supported_devices.html.

Even if one of your devices is not on the list of supported devices, chances are that you can easily add support for it. The procedure to add support of a new device is extensively documented on https://www.zigbee2mqtt.io/how_tos/how_to_support_new_devices.html.

While Zigbee2mqtt does the heavy lifting, it doesn't have a web interface. That's why you also should install Zigbee2MqttAssistant (https://github.com/yllibed/Zigbee2MqttAssistant). It displays all your Zigbee devices and their status in a web interface and lets you configure, rename, and remove your devices. It's also able to display an interactive map of your Zigbee network.

Warning:
Zigbee2MqttAssistant doesn't support HTTPS and also doesn't have an option to protect its web interface with a password, so you should only visit it on a trusted network. To add HTTPS and authentication anyway, you can add a Docker container running nginx to your Docker Compose file and configure it to run as a reverse proxy with HTTPS and authentication for your Zigbee2MqttAssistant container. See the appendix at the end of this book for a way to do this.

9.3.1 ● Connecting the CC2531

With the correct firmware on your CC2531 now, you can connect it to your Raspberry Pi again, but it's recommended to do this not directly, but with a USB extension cable. Even a short one of 20 cm to 50 cm long is already enough to put the CC2531 further from the on-board Wi-Fi and Bluetooth antennas of the Raspberry Pi and reduce interference.

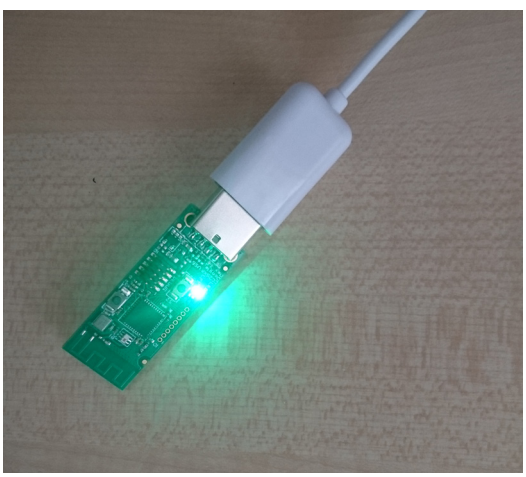

Figure 9.4 Connect the CC2531 with an USB extension cable to reduce interference from the Raspberry Pi's on-board Wi-Fi and Bluetooth.

9.3.2 ● Installing Zigbee2mqtt and Zigbee2MqttAssistant

Just like almost all the software in this book, you'll run Zigbee2mqtt and Zigbee2MqttAssistant in Docker containers. First create a directory for Zigbee2mqtt to store its data in:

```
mkdir -p /home/pi/containers/zigbee2mqtt
```

Zigbee2MqttAssistant doesn't need a configuration file, as you configure it with environment variables.

Then edit your docker-compose.yml file in your home directory with the following content:

```
version: '3.7'

services:
  mosquitto:
    # mosquitto configuration
  zigbee2mqtt:
    image: koenkk/zigbee2mqtt
    container_name: zigbee2mqtt
    volumes:
      - ./containers/zigbee2mqtt:/app/data
    devices:
      - /dev/ttyACM0:/dev/ttyACM0
    restart: always
    environment:
      - TZ=Europe/Brussels
  zigbee2mqttassistant:
    image: carldebilly/zigbee2mqttassistant
    container_name: zigbee2mqttassistant
    environment:
      - Z2MA_SETTINGS__MQTTSERVER=mosquitto
      - Z2MA_SETTINGS__MQTTUSERNAME=home
      - Z2MA_SETTINGS__MQTTPASSWORD=PASSWORD
      - TZ=Europe/Brussels
    ports:
      - 8880:80
    restart: always
```

Some of these values should be adapted to your situation. For instance, the TZ environment variable should be set to your location's time zone, and the device should have the correct path. Normally you should see the right path for your setup if you execute the command

`dmesg | grep tty` after you have connected the CC2531 to the USB port. See the appendix at the end of the book for a better way using the device ID.

For the zigbee2mqttassistant container, enter the password you created for the user `home` in your Mosquitto configuration after the `Z2MA_SETTINGS__MQTTPASSWORD` environment variable.

Note:
If you want to run Zigbee2MqttAssistant on another device than the one running your MQTT broker, you're currently out of luck: Zigbee2MqttAssistant doesn't support (yet) certificates signed by your own CA. I reported this issue on https://github.com/yllibed/Zigbee2MqttAssistant/issues/251, and hopefully by the time you read this book you can set the CA file with an environment variable.

Now start the container:

```
docker-compose up -d
```

This will take a while because Docker downloads and then starts both containers.

9.3.3 ● Configuring Zigbee2mqtt and Zigbee2MqttAssistant

While Zigbee2MqttAssistant is configured with environment variables (see https://github.com/yllibed/Zigbee2MqttAssistant for all the variables you can configure), Zigbee2mqtt uses a file, `configuration.yaml`. It's been created automatically the first time the Zigbee2mqtt container starts. So let's first stop the container:

```
docker-compose stop zigbee2mqtt
```

Open the configuration file in an editor:

```
sudo nano /home/pi/containers/zigbee2mqtt/configuration.yaml
```

Change the MQTT section to the following:

```
# MQTT settings
mqtt:
  # MQTT base topic for zigbee2mqtt MQTT messages
  base_topic: zigbee2mqtt
  # MQTT server URL
  server: 'mqtt://mosquitto'
  # MQTT server authentication
  user: home
  password: PASSWORD
```

This supposes that you're running Zigbee2mqtt on the same Raspberry Pi as your MQTT broker. Make sure you enter the correct password for the `home` user on your MQTT broker.

Note:
If you want to run Zigbee2mqtt on another device than the one running your MQTT broker, look at the configuration example on https://www.zigbee2mqtt.io/information/configuration.html. The server URL should be 'mqtts://HOSTNAME' with the fully qualified hostname of your MQTT broker, and you should add the path to your trusted CA in the ca variable. Also, don't forget to mount a volume backed by your TLS certificates directory for the zigbee2mqtt service in your Docker Compose file.

By default, Zigbee2mqtt uses a well-known key for network encryption. It's recommended to use a different one. A network encryption key is a 128-bit number, which is essentially 16 decimal values between 0 and 255. You can create a random network encryption key with the following command:

```
dd if=/dev/urandom bs=1 count=16 2>/dev/null | od -A n -t u1 | awk '{printf ⏎
"["} {for(i = 1; i< NF; i++) {printf "%s, ", $i}} {printf "%s]\n", $NF}'
```

Copy the output of this command, and paste it in the right section in your `configuration.yaml`, like this:

```
advanced:
  network_key: [133, 242, 71, 182, 54, 45, 233, 255, 23, 38, 194, 62, 62, ⏎
230, 255, 48]
```

You can change a lot more in the configuration file. Have a look at https://www.zigbee2mqtt.io/information/configuration.html for all available options, including experimental ones.

Note:
One interesting change could be to add the `line disable_led: true` in the `serial:` section because the CC2531's green LED is very bright.

After entering the correct settings, restart the container with:

```
docker-compose start zigbee2mqtt
```

And then check whether it has started correctly:

```
docker-compose logs -f zigbee2mqtt
```

If all goes well, you should only see green (info) and some yellow (warn) messages. One of the info messages that Zigbee2mqtt shows is the firmware version on the CC2531. If you see a red message (error), try to find a solution to the problem. Afterwards, restart the container if needed.

9.3.4 ● Using Zigbee2MqttAssistant

After the initial configuration, you'll mostly use Zigbee2MqttAssistant and not Zigbee2mqtt directly. You can access it in your web browser on http://IP:8880, with the IP address of your Raspberry Pi. When you don't have paired any devices yet, you'll see something like this:

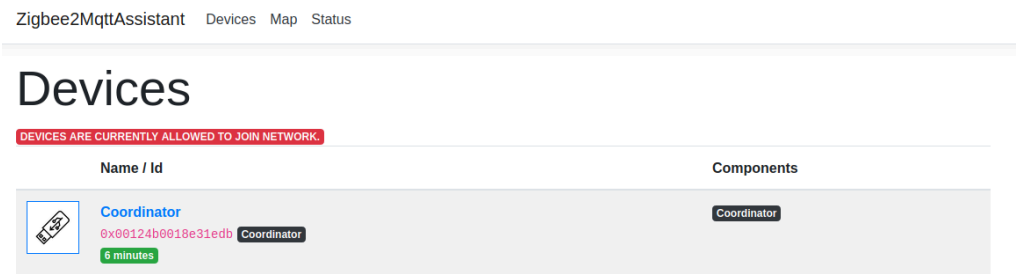

Figure 9.5 Zigbee2MqttAssistant shows your CC2531 as the coordinator of your Zigbee network.

If you don't see your coordinator, have a closer look at the logs, because this means that your CC2531 hasn't been detected.

You can also verify whether your setup is working by clicking on **Status**. At the top of the page, you should see **Bridge State: online**.

Now the next step is to pair your Zigbee devices. For this, your Zigbee coordinator should allow other devices to join the network. While Zigbee2mqtt allows this in its default configuration (with the line `permit_join: true` in its `configuration.yaml`), Zigbee2MqttAssistant automatically denies new devices to join the network after 20 minutes, even if Zigbee2mqtt's configuration allows it.

If Zigbee2MqttAssistant has been started more than 20 minutes ago, you can always allow pairing temporarily by clicking on **Allow new devices to join the network** at the top of the Status tab. You have 20 minutes now to pair your Zigbee devices. You can always deny new pairing requests after you're done, by clicking on **Deny new devices to join the network** in the same tab of Zigbee2MqttAssistant's web interface.

The procedure to pair your Zigbee devices depends on your specific device, and you should consult its user manual for more details. However, Zigbee2mqtt has also extensive documentation about each supported device, including pairing instructions, recommendations, and tips for its configuration. You should search all your devices on the https://www.zigbee2mqtt.io/information/supported_devices.html page and consult each device's page.

For instance, for the Xiaomi Aqara door and window contact sensor (Xiaomi MCCGQ11LM), the pairing procedure consists of pressing and holding the reset button on the device for 5 seconds. When the blue light starts blinking, you can stop pressing and the device is paired to your Zigbee coordinator.

If you look at the **Devices** tab now (you may need to refresh the web page), a new device has appeared. Zigbee2MqttAssistant even shows a picture of the device, and in the right column, it shows some 'components' of the device. For instance, for a door and window contact sensor the components are `battery`, `contact`, and `linkquality`.

The first column shows the numerical ID of the device, as well as a friendly name. By default, the friendly name for all newly added devices is the same as its numerical ID. So click on the name, and then this device-specific page allows you to change its name to a more user-friendly one. Click on **Rename**. After this, you should see your configured name as the friendly name in the device list. Note that to prevent some issues, you should restart Zigbee2mqtt after every name change. This can be done with:

```
docker-compose restart zigbee2mqtt
```

Note:
In contrast to Z-Wave devices, there's very little you can configure in the way your Zigbee devices function.

9.4 • Using our Zigbee devices with MQTT

After you have paired and configured your Zigbee devices and everything works, you shouldn't use the web interface of Zigbee2MqttAssistant that often anymore. You shouldn't even have to know about Zigbee2mqtt. Both services are there if you need to check something (the logs are often helpful when you encounter a problem with your Zigbee network), but in most cases you just interact with your Zigbee network using MQTT. The MQTT topics and message structures are nicely documented on https://www.zigbee2mqtt.io/information/mqtt_topics_and_message_structure.html.

The first thing you can do is have a look at what messages Zigbee2mqtt emits. Try this `mosquitto_sub` command:

```
mosquitto_sub -t 'zigbee2mqtt/#' -v
```

Use the same base topic (`zigbee2mqtt`) as the one you have configured in the base_topic key in the file `configuration.yaml` in Zigbee2mqtt's directory.

Even if nothing is going on in your Zigbee network at the moment, this command should show you at least the configuration of your coordinator and optionally the configuration of some of your devices. For instance, the configuration of Zigbee2mqtt and its coordinator is sent to the `zigbee2mqtt/bridge/config` topic as a JSON string. In a more human-readable representation with indents this should look like:

```
{
  "version": "1.11.0",
  "commit": "31e5678",
  "coordinator": {
    "type": "zStack12",
    "meta": {
      "transportrev": 2,
      "product": 0,
      "majorrel": 2,
      "minorrel": 6,
      "maintrel": 3,
      "revision": 20190608
    }
  },
  "log_level": "info",
  "permit_join": true
}
```

9.4.1 ● Reading sensor values

As soon as one of your Zigbee sensors emits a value, you should see a new MQTT message, such as:

```
zigbee2mqtt/Cabinet door
{"battery":100,"voltage":3055,"contact":false,"linkquality":52}
```

In this case, I opened the cabinet door that has the Xiaomi Aqara door and window contact sensor attached. This output of `mosquitto_sub` first shows the topic: `zigbee2mqtt/Cabinet door`. Every device in your Zigbee network gets a subtopic on a level below the base topic `zigbee2mqtt`, and its subtopic is just the friendly name that you have configured for the device in Zigbee2MqttAssistant, and otherwise its ID.

The message itself is in JSON format again, and it shows the components of the sensor as values of its keys. For this Xiaomi Aqara door and window contact sensor, this looks like this (if I indent the JSON code for more readability):

```
{
  "battery": 100,
  "voltage": 3055,
  "contact": false,
  "linkquality": 52
}
```

In this case, you see that the device sends its battery percentage, voltage, contact (`true` for a closed door, `false` for an open door) and link quality (from 0 to 255, with 255 the best link quality).

In this case, the piece of information you are interested in is the state of the contact: the door has opened. If you want to use this value in your program, you can easily subscribe to the relevant MQTT topics with the Paho MQTT Python library, parse the JSON content of the messages and react to the values.

9.4.2 ● Controlling switches

Now how do you set a value, for instance, if you want to turn on a switch or lamp? For this, you send an MQTT message to the MQTT topic `zigbee2mqtt/FRIENDLY NAME/set` with a JSON payload. For instance, for the Xiaomi Mi power plug (ZNCZ02LM) the command to switch the power plug on becomes:

```
mosquitto_pub -t 'zigbee2mqtt/coffee/set' -m '{"state": "ON"}'
```

Zigbee2mqtt now receives your command, translates it to a Zigbee command, and sends it to the right device. As a result, the power plug turns on. You can find the available keys in the JSON payload on the page for the device on Zigbee2mqtt's web site.

If you don't like the verbosity of the JSON payload, you can also publish on a subtopic with the `state` key. The payload then just becomes a string with the value of the key:

```
mosquitto_pub -t 'zigbee2mqtt/coffee/set/state' -m 'ON'
```

If a switch supports this, you can also read the current state of the device by sending this command:

```
mosquitto_pub -t 'zigbee2mqtt/coffee/get' -m '{"state": ""}'
```

9.5 ● Summary and further exploration

Zigbee is a popular wireless home automation protocol, and in this chapter, I showed you how you can add a Zigbee transceiver to turn your Raspberry Pi into a Zigbee coordinator, so you don't need another Zigbee gateway such as the Philips Hue Bridge anymore.
The magic behind all this is Zigbee2mqtt, which lets you interact with your Zigbee devices using MQTT. This software essentially translates Zigbee to MQTT and vice versa. I also added Zigbee2MqttAssistant, which is a simple web interface for Zigbee2mqtt. After this, you can read sensor values by subscribing to MQTT topics and control switches and lights by publishing to MQTT topics.

Zigbee2mqtt is not the only way to integrate Zigbee with your home automation gateway. Another project that implements the Zigbee protocol stack is zigpy (https://github.com/zigpy/zigpy). It supports a lot more Zigbee transceivers than the CC2531 family, including the popular ConBee II and the HUSBZB-1 (which supports both Zigbee and Z-Wave in one USB transceiver). Zigpy is the project that is used in Home Assistant (see Chapter 10) to support Zigbee devices. If you want to focus more on Home Assistant, this is an interesting project to explore.

Chapter 10 ● Automating your home

In the previous chapters, I've explained a lot of home automation protocols and how you could link them to MQTT. If you have tried the software I showed, you are already on the right track to home automation, but you can barely call it automation. What you have done until now is more or less limited to message passing: one piece of software reacts to an event (such as receiving a sensor value) by publishing a message to another piece of software (in most cases an MQTT broker).

That's why in this chapter I'll show you how you can automate your devices. You'll learn how to:

- create dashboards to show you current and historical sensor values;
- alert you when your fridge is too warm;
- alert you when your garage door is open.

You can automate your home perfectly by writing Python scripts using the Eclipse Paho MQTT client library (see Chapter 4), which I already showed you in a couple of chapters. But that won't result in very maintainable code. You need a home automation platform that takes away some of the bookkeeping and boilerplate code, such as connecting to your MQTT broker, registering callback functions, and so on.

Luckily there are a lot of open-source home automation platforms, and I'll cover three of them in this chapter and show you how to automate your home with them. However, I won't go into too much detail, because these platforms are vast and have a lot of functionality. You could write a whole book about each one of them.[1] Many of them are also in constant flux with fast developments, so by the time you read this book, a lot of the advanced functionality has probably already changed. But I'll give you an overview of the basic functionality in each of these platforms. You can choose the one you prefer, or even use more than one simultaneously.

In this chapter I cover the following systems, that each have their strengths:

Node-RED
A 'low-code' platform that lets you 'wire together' devices, APIs, and online services in a graphical way, so you don't have to write code.

Home Assistant
A popular home automation gateway that puts local control and privacy first, with integrations to a lot of home automation devices and protocols, as well as a user-friendly way to add automations.

1 For instance, if you want to delve more into Node-RED, have a look at the book Programming with Node-RED: Design IoT Projects with Raspberry Pi, Arduino and ESP32 (https://www.elektor.com/programming-with-node-red) by Dogan Ibrahim.

AppDaemon
A multithreaded Python execution environment for writing apps for home automation projects. It has direct support for Home Assistant and MQTT.

It's clear that each of them has a specific focus, so it's not possible to call one of them the 'best'. I'll show you some of their strengths in the next sections.

> **Note:**
> There are many home automation platforms. The fact that I don't cover them in this book doesn't mean that they are bad choices. If you're already using another home automation platform, by all means, keep using it, and try to find how you can integrate it with the projects covered in this book. And if the platforms in this chapter are just not your cup of tea, you can explore some of the alternatives I list at the end of this chapter.

10.1 ● Node-RED

Node-RED (https://nodered.org) is a 'low-code' platform that lets you 'wire together' devices, APIs, and online services in a graphical way, so you don't have to write code. It's completely web-based, so you can create your automations in a web browser.

You create your automations by dragging components ("nodes") to the canvas and then connecting them. Under the hood, Node-RED is running JavaScript functions on the Node.js platform (https://nodejs.org).

10.1.1 ● Installing Node-RED

First, create a directory where Node-RED can store its data:

```
mkdir /home/pi/containers/node-red
```

Add a Node-RED container to your docker-compose.yml file:

```
version: '3.7'

services:
  mosquitto:
    # mosquitto configuration
  node-red:
    image: nodered/node-red
    container_name: node-red
    restart: always
    volumes:
```

```
      - ./containers/node-red:/data
      - ./containers/certificates:/etc/ssl/private:ro
      - /etc/localtime:/etc/localtime:ro

    ports:
      - "1880:1880"
```

Then start Node-RED:

```
docker-compose up -d
```

After this, you can visit http://IP:1880, with IP the IP address of your Raspberry Pi, to view Node-RED's web interface. However, this doesn't use HTTPS and isn't protected with a password. You should start Node-RED one time with this default configuration so it creates a configuration file.

10.1.2 ● Adding authentication to Node-RED

Now that the first run of the Node-RED container has created its configuration file, you can change this to add authentication. This can be done for three places in Node-RED:

- the editor;
- the dashboard;
- static pages.

You can enter the passwords there as a password hash. Generating a password hash can be performed in the Node-RED container, for instance using:

```
docker exec -ti node-red /usr/local/bin/node -e "console.⏎
log(require('bcryptjs').hashSync(process.argv[1], 8));" PASSWORD
```

Just replace `PASSWORD` with a password of your choice. The result is a long string such as `$2a$08$eNu717MuUoucCgUAWKM9Qeymf94ps6qxaJHTzdA2hDOvy2pbrYH12.`

Now open Node-RED's configuration file:

```
nano /home/pi/containers/node-red/settings.js
```

Almost all configuration options are commented out with two slashes (`//`). Scroll to the following section:

```
    // Securing Node-RED
    // -----------------
    // To password protect the Node-RED editor and admin API, the following
    // property can be used. See http://nodered.org/docs/security.html for
details.
    //adminAuth: {
    //    type: "credentials",
    //    users: [{
    //        username: "admin",
    //        password:
"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN.",
    //        permissions: "*"
    //    }]
    //},

   // To password protect the node-defined HTTP endpoints (httpNodeRoot), or
    // the static content (httpStatic), the following properties can be used.
    // The pass field is a bcrypt hash of the password.
    // See http://nodered.org/docs/security.html#generating-the-password-hash
    //httpNodeAuth

{user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9D
N."},
    //httpStaticAuth:
{user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9D
N."},
```

And change it to something like this:

```
    // Securing Node-RED
    // -----------------
    // To password protect the Node-RED editor and admin API, the following
    // property can be used. See http://nodered.org/docs/security.html for
details.
    adminAuth: {
        type: "credentials",
        users: [{
            username: "admin",
            password:
"$2a$08$eNu717MuUoucCgUAWKM9Qeymf94ps6qxaJHTzdA2hDOvy2pbrYH12",
            permissions: "*"
        }]
    },

    // To password protect the node-defined HTTP endpoints (httpNodeRoot), or
```

```
    // the static content (httpStatic), the following properties can be used.
    // The pass field is a bcrypt hash of the password.
    // See http://nodered.org/docs/security.html#generating-the-password-hash
    httpNodeAuth:
{user:"user",pass:"$2a$08$eNu717MuUoucCgUAWKM9Qeymf94ps6qxaJHTzdA2hDOvy2pbrYH
12"},
    httpStaticAuth:
{user:"user",pass:"$2a$08$eNu717MuUoucCgUAWKM9Qeymf94ps6qxaJHTzdA2hDOvy2pbrYH
12"},
```

Make sure to enter the password hash you generated in the place of the `password` string for the admin authentication (for the Node-Red editor) and enter the same or another password hash in the place of the `pass` string for the node (dashboard) and static pages. You can also change the default usernames (`admin` in `adminAuth` and user in `httpNodeAuth` and `httpStaticAuth`) if you want.

After these changes, restart the Node-RED container:

```
docker-compose restart node-red
```

If you now reopen **http://IP:1880** in your web browser, you first have to enter the right username and password before you're allowed in the flow editor.

10.1.3 ● Using Node-RED over HTTPS

There's still one task to do to secure your Node-RED setup: using HTTPS instead of HTTP. You have already mounted a volume in the container with your Raspberry Pi's certificate files in the Docker Compose file, so you just have to refer to these files in Node-RED's configuration. Reopen the configuration file. At the beginning, uncomment the line that begins with `//var fs = require("fs");`, so it looks like this:

```
var fs = require("fs");
```

Then scroll further down again to the section beginning with `// Securing Node-RED`. After this section, you see these lines:

```
    // The following property can be used to enable HTTPS
   // See http://nodejs.org/api/https.html#https_https_createserver_options_
requestlistener
   // for details on its contents.
```

```
    // See the comment at the top of this file on how to load the `fs` module
used by
    // this setting.
    //
    //https: {
    //    key: fs.readFileSync('privatekey.pem'),
    //    cert: fs.readFileSync('certificate.pem')
    //},

    // The following property can be used to cause insecure HTTP connections
to
    // be redirected to HTTPS.
    //requireHttps: true,

   // for details on its contents.
```

Change them to something like this:

```
    // The following property can be used to enable HTTPS
    // See http://nodejs.org/api/https.html#https_https_createserver_
options_requestlistener
    // for details on its contents.
    // See the comment at the top of this file on how to load the `fs`
module used by
    // this setting.
    //
    https: {
        key: fs.readFileSync('/etc/ssl/private/key.pem'),
        cert: fs.readFileSync('/etc/ssl/private/cert.pem')
    },

    // The following property can be used to cause insecure HTTP
connections to
    // be redirected to HTTPS.
    requireHttps: true,
```

Make sure to adapt the filenames of the key and certificate to your situation.

After these changes, restart the Node-RED container:

```
docker-compose restart node-red
```

Your Node-RED installation is now not accessible anymore using HTTP. But if you enter **https://HOSTNAME:1880** in your web browser, you're asked the username and password for the flow editor. Your Node-RED installation is now secure and all communication is encrypted.

10.1.4 ● Creating Node-RED flows

In Node-RED you don't type code to create a program, but you connect nodes to create a flow. In the sidebar at the left, you see a lot of nodes, arranged in categories such as **common**, **function**, **network**, **sequence**, **parser** and **storage**.

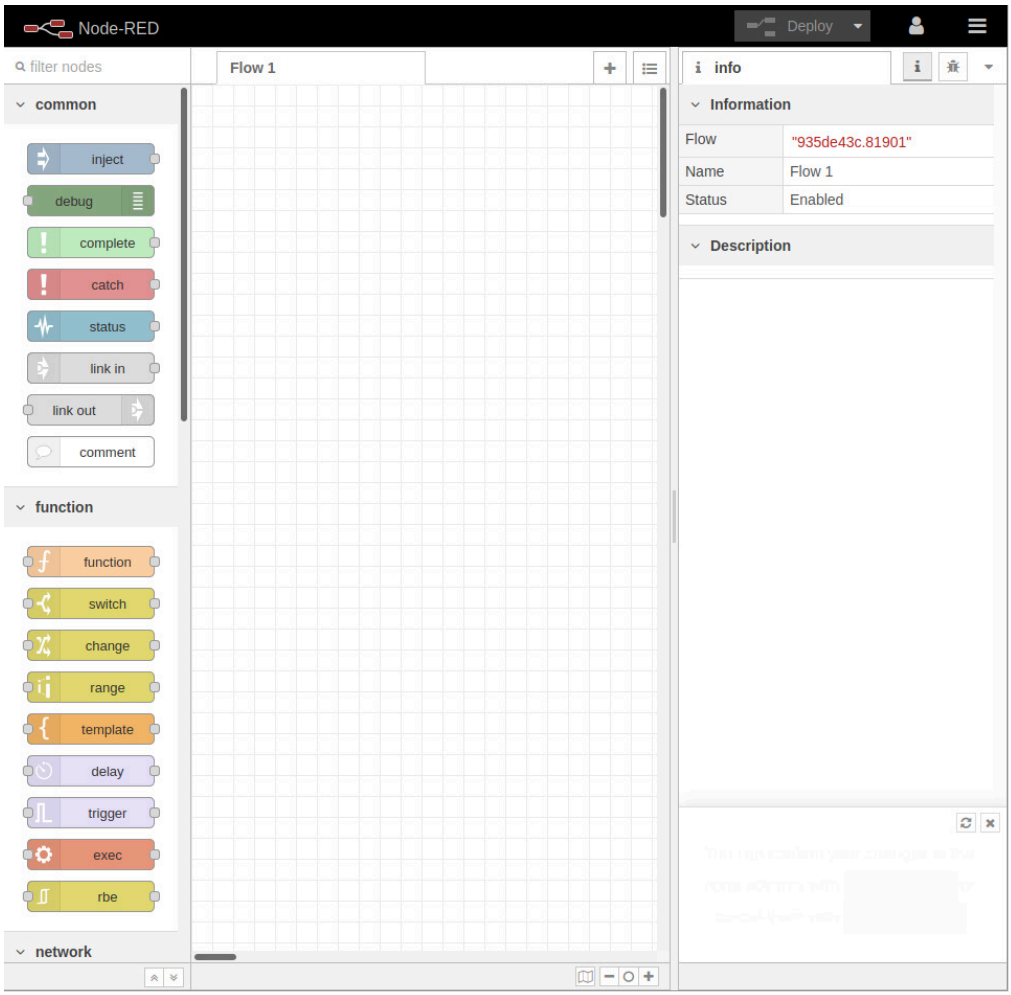

Figure 10.1 Node-RED shows an empty canvas where you can put nodes on to create a flow.

For instance, let's revisit the Python program in Chapter 4 that sends an MQTT message

with the time to your broker when someone asks for it. Rewriting it in the form of a Node-RED flow is a good way to familiarize yourself with the Node-RED platform.

Note:
You can add more than one flow to Node-RED: just click on the + icon at the right above the canvas. You can also change the name of a flow by double-clicking on the current name (by default **Flow 1**) and then editing the **Name** field.

First, drag the **mqtt in** node in the category **network** to the canvas. This node lets you react to an MQTT message with a specific topic. Double-click on the node on your canvas to edit its properties.

Now you have to define the MQTT broker this node is listening to. Next to **Server**, select **Add new mqtt-broker...** and click on the pencil on the right.

Give this MQTT broker a name and add its hostname in the **Server** text field. If you're running Mosquitto as a broker in a Docker container defined in the same Docker Compose file as Node-RED at the beginning of this chapter, you can just enter **mosquitto** here, which is the name of the Docker container Mosquitto's running in.

If you're using an MQTT broker on another machine, you should have configured TLS for it (see Chapter 4) and you should choose port 8883 here. Make sure that the hostname is the same one as the one the TLS certificate is created for. Also, click on **Enable secure (SSL/TLS) connection**. A new dropdown list appears then: **TLS Configuration**. Click on the pencil icon next to **Add next tls-config...** and then upload the CA certificate for your MQTT broker (`rootCA.pem`, see Chapter 3) next to **CA Certificate**. You can ignore the other files.[2] Give the TLS configuration a name and click **Add**.

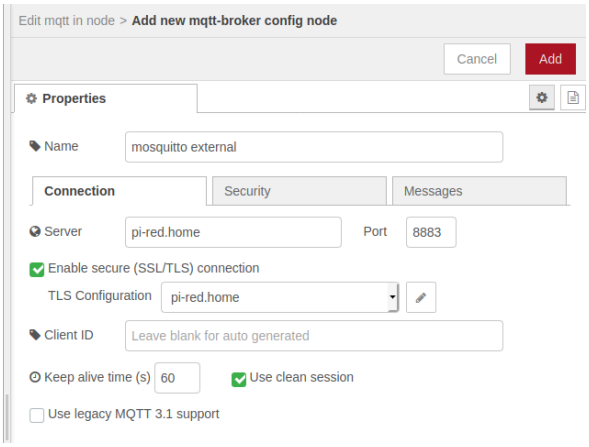

Figure 10.2 Node-RED is able to communicate with an MQTT broker.

2 Those are for a client certificate, which is out of scope for this book. A client certificate can be used to authenticate to the MQTT broker instead of a username and password.

After this, you're back in the properties of the **mqtt in** node. Go to the **Security** tab and supply a valid username and password. Finally, click on **Add** to add this broker configuration.

The broker you just configured is now defined as this node's broker. Next, choose the topic this node is listening to: `time/request`. And optionally give the node a name, such as **Time request**. Click on **Done** to save the node.

Now drag the **mqtt out** node to the canvas, and double-click on it to edit its properties. Choose the same broker as the server, and enter `time/reply` as the topic. Optionally give the node a name, such as **Time reply**, and click **Done** to save it.

Then link both nodes (just drag a line between the little square at the right of the first node and the square at the left of the second node), and click on **Deploy** at the top right. Both nodes should now show a green block below with the message **connected**. This means that the nodes are successfully connected to the MQTT broker.

This flow now does something very simple: if Node-RED receives a message on MQTT topic `time/request`, it echoes the message to another topic, `time/reply`.

Verify this on the command line with the `mosquitto_sub` and `mosquitto_pub` commands. Open a shell and subscribe to the `time/reply` topic:

```
mosquitto_sub -t 'time/reply' -v
```

In another shell, send a request:

```
mosquitto_pub -t 'time/request' -m ''
```

Now you should see **time/reply (null)** in the first shell, which is an empty payload. Of course, this isn't that useful, so let's change the flow so it replies the current time. Add a **function** node (in the category **function**), delete the existing connection between the two MQTT nodes, and link the output of the time request to the input of the function node and the output of the function node to the input of the time reply.

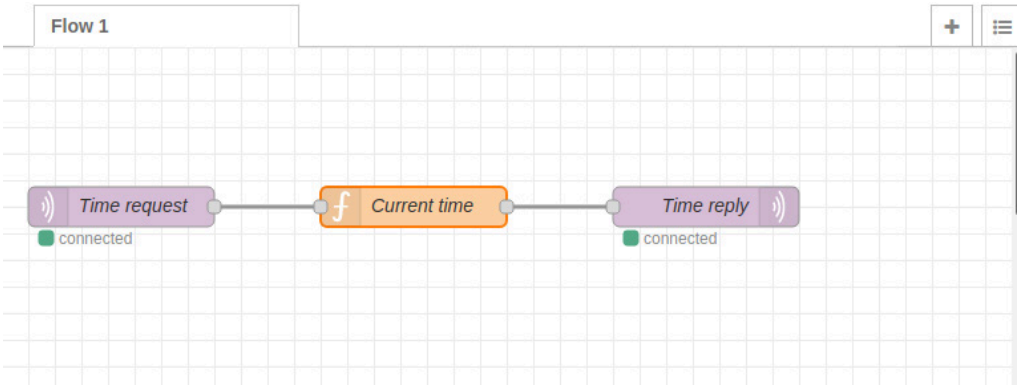

Figure 10.3 This simple Node-RED flow sends the current time as an MQTT message in reply to another message.

Double-click on the function node, enter a name such as **Current time** and enter the following JavaScript code in the **Function** text field:

```
var date = new Date();
msg.payload = date.toString();
return msg;
```

This creates a Date object in JavaScript and sets the payload of the MQTT message to a string representation of this date. Click on **Done** to save the node and don't forget to click on **Deploy** to deploy your changes. Now send an MQTT message again on the **time/request** topic and look at the message you get on `time/reply`:

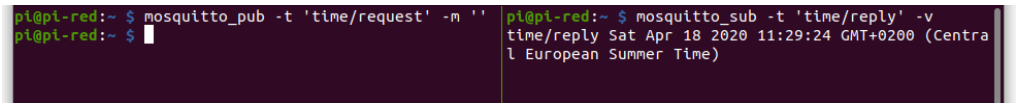

Figure 10.4 This Node-RED flow sends the `time/reply` MQTT message with the current time as its payload.

Note:
If you add a **debug** node to the canvas and connect it to the output of another node, you can see the payload of the node in a debug panel. The debug panel is shown at the right when you open the hamburger menu and then **View > Debug messages.**

10.1.5 ● Installing extra nodes in Node-RED

Node-RED has a big ecosystem of external nodes you can install. Just open the hamburger menu (the three horizontal lines) at the top right and then choose **Manage palette**. If you click on the **Install** tab, you get access to thousands of nodes.

Just start typing some keyword and the shown nodes are filtered. For instance, let's install the **node-red-contrib-moment** node: search for **moment** and then click on **install** next to the right node and then **Install** again.

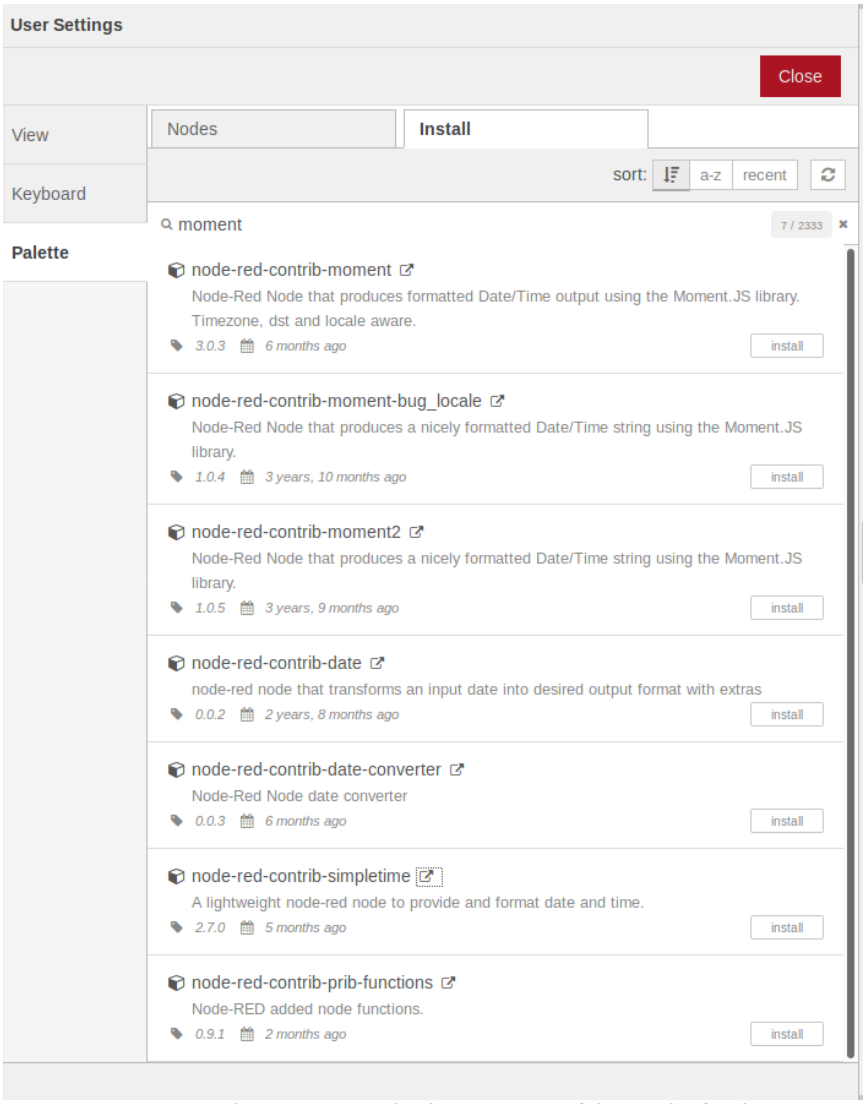

Figure 10.5 Node-RED comes with a big ecosystem of thousands of nodes.

After installation, you can close the palette manager. Two new node types have been added to your pallette: **moment** and **humanizer**. Let's use the **moment** node to improve on the flow from the previous subsection and show a more readable time.

Remove the connection between the **function** node and the **mqtt out** node from the previous subsection, drag the **moment** node between them, and recreate the connections.

Now edit the properties of the **moment** node. Put your timezone right, and enter **YYYY-MM-DD HH:mm** in the **Output Format** field. After deploying your Node-RED flow, an MQTT message to the `time/request` topic gives you a message with a time such as **2019-11-30 17:56** to the `time/reply` topic.

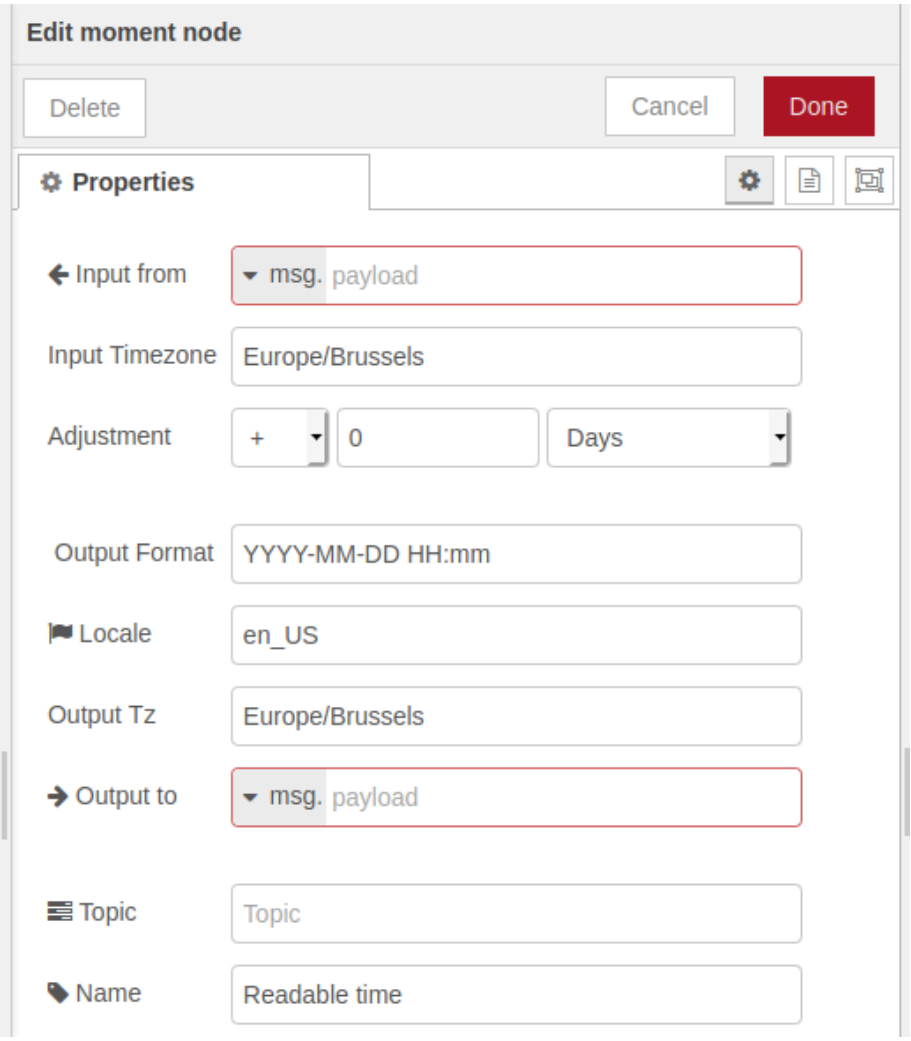

Figure 10.6 The moment node lets you use a more readable output format for dates and times.

There's still a lot more to say about flows in Node-RED, but I'll give a slightly more complex example in Chapter 12, so this will suffice for now.

10.1.6 ● Creating a dashboard in Node-RED

The flow you made in the previous subsections just uses MQTT for its output, but Node-RED also has a nice way to create a user interface: dashboards. Let's create a dashboard that shows sensor readings that are sent as MQTT messages.

The Node-RED dashboard isn't installed by default, so return to **Manage palette** in the hamburger menu and install **node-red-dashboard**. You'll see a message that a lot of new nodes have been installed. Close the settings after installation.

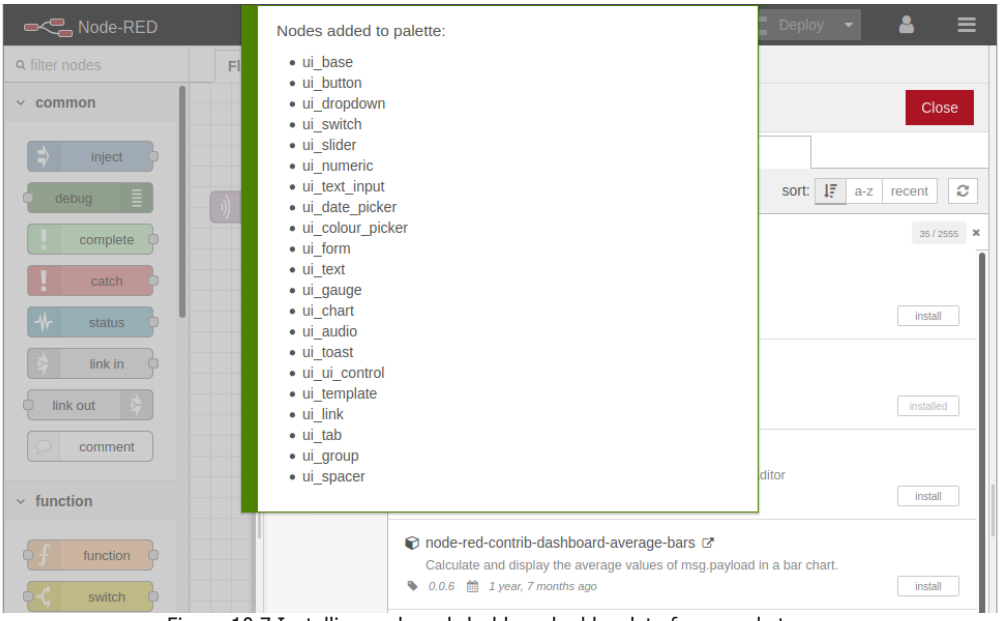

Figure 10.7 Installing node-red-dashboard adds a lot of new node types.

Now click on the plus sign at the top right to create a new flow. Double-click on its name (probably **Flow 2**) and change the name to something more descriptive.

Let's first add an **mqtt in** node to read your sensor data. I'll use the sensor data of a RuuviTag as an example, published as MQTT messages by bt-mqtt-gateway (see Chapter 6). If you want to use something else, just change the MQTT topic. So in this case, I enter `bt-mqtt-gateway/ruuvitag/bedroom/temperature` as the topic in the properties of the **mqtt in** node. Give the node a name (such as **Bedroom temperature**) and make sure you point it to the correct MQTT broker:

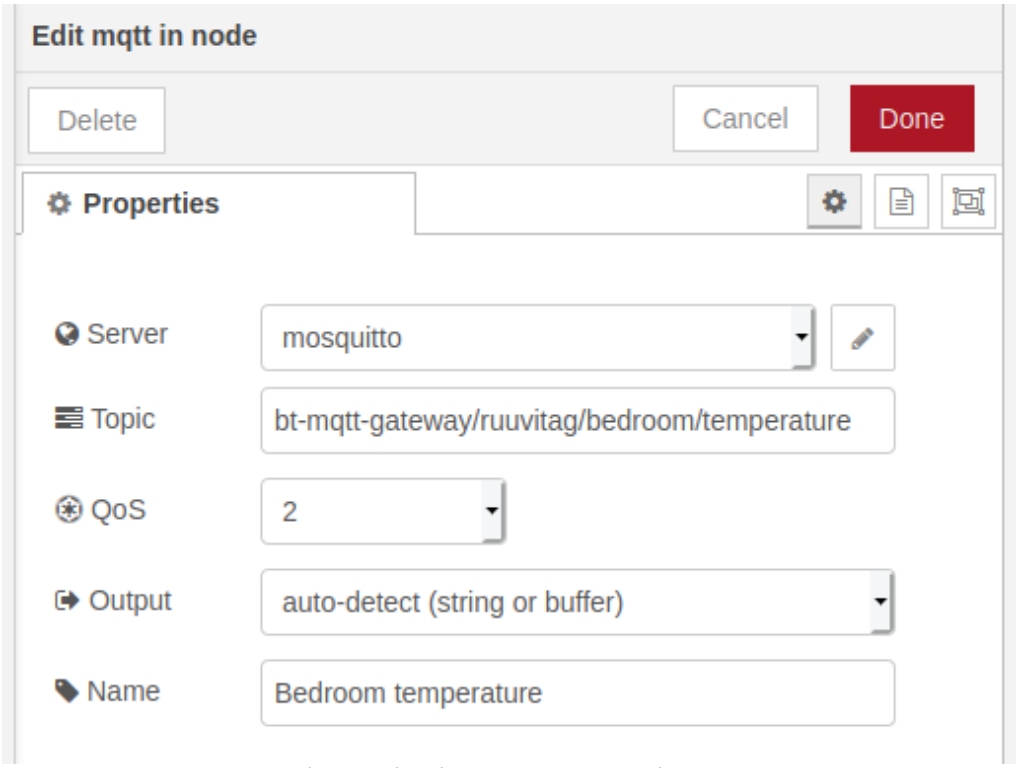

Figure 10.8 Let Node-RED subscribe to an MQTT topic with sensor measurements.

Now add an **mqtt in** node for every sensor measurement you want to show in your dashboard. After this, scroll at the bottom of your palette at the left sidebar and drag a **gauge** node (from the **dashboard** collection of nodes) after every **mqtt in** node. Then connect each **mqtt in** node with their corresponding gauge node.

There's an orange triangle on top of each of the gauge nodes, which means that their configuration is invalid or incomplete. So double-click on one of the gauges. Next to Group, you see **Add new ui_group** with a red outline. This is the incomplete configuration that the orange triangle warned you about. Click on the pencil icon next to it. This creates a new dashboard group, which groups some dashboard widgets. Enter the name of your sensor, for instance, **Bedroom**, for the **Name field**.

This node also needs you to choose a **Tab**. So click on the pencil right to **Add new ui_tab...** to create a new tab. I'll use a tab for each sensor. So give this tab a name such as **Environmental sensors** and optionally the name of an icon.[3] Then click **Add** to create the tab. After this, you return to the group properties and click again on **Add** to create the

3 You can search for available icon descriptions on the websites of Material Design Icons (https://materialdesignicons.com), Font Awesome (https://fontawesome.com) and Weather Icons (https://github.com/Paul-Reed/weather-icons-lite/blob/master/css_mappings.md).

group.

Now you return to the gauge properties. Change the **Label** field to **Temperature** for a temperature sensor, choose your unit, and enter a range with minimal and maximal values you expect for this sensor. You can also add threshold values for the color gradients, and you can even change the colors, for instance, blue for cold temperatures, green for normal temperatures, and red for hot temperatures. Finally, click on **Done**.

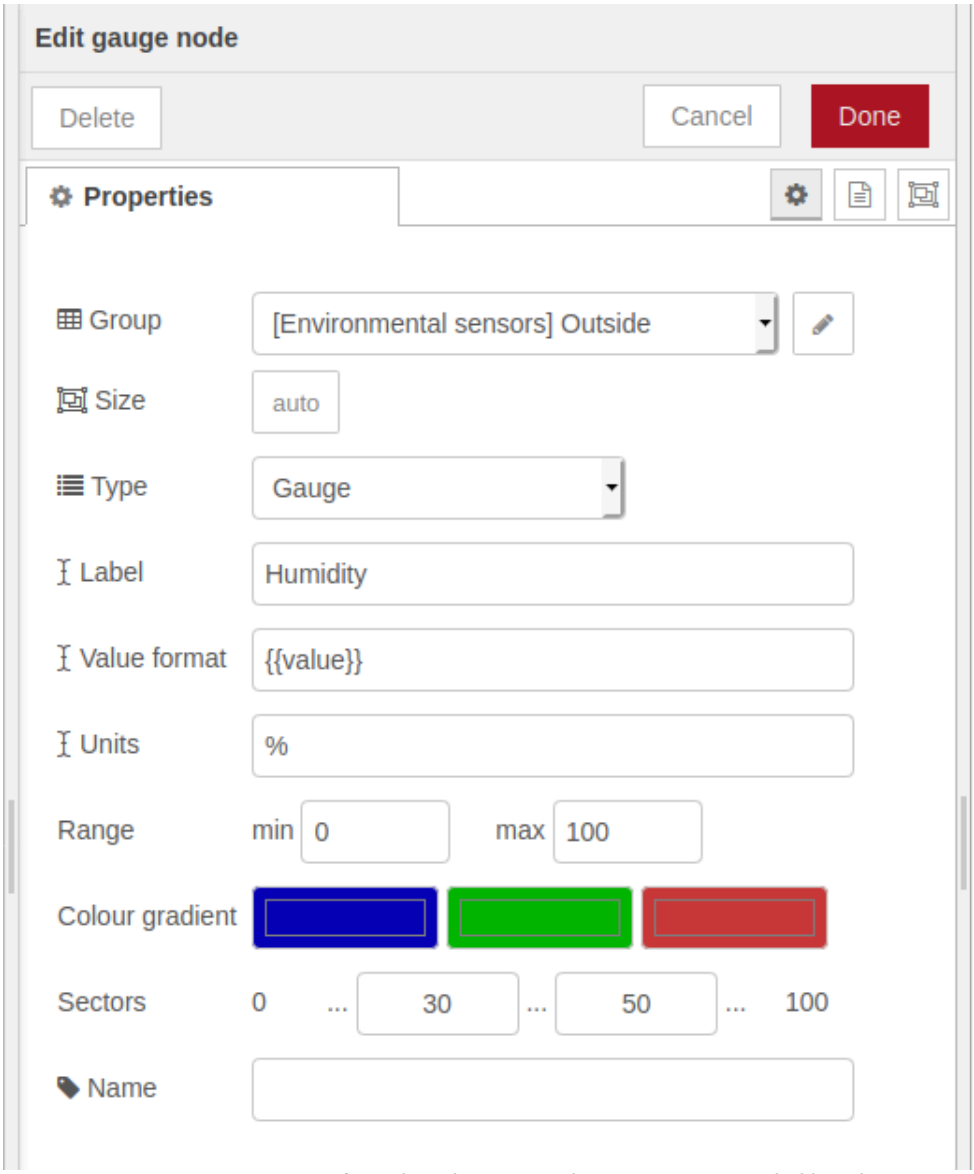

Figure 10.9 You can configure how the gauge widget appears on your dashboard.

Now do the same for all your other gauge nodes. For instance, for the node showing the humidity of the same sensor that you just added the temperature for, you can reuse the same group. Just change the other properties, such as the units, range, colour gradients, and thresholds.

For other sensors, click on the dropdown list next to **Group** and select **Add new ui_group** and then click on the pencil. Give it the name of the sensor, make sure that **Tab** still lists the tab you have created and then click on **Add** to create this new group. Then for other gauge nodes for the same sensor choose this group in the dropdown list.

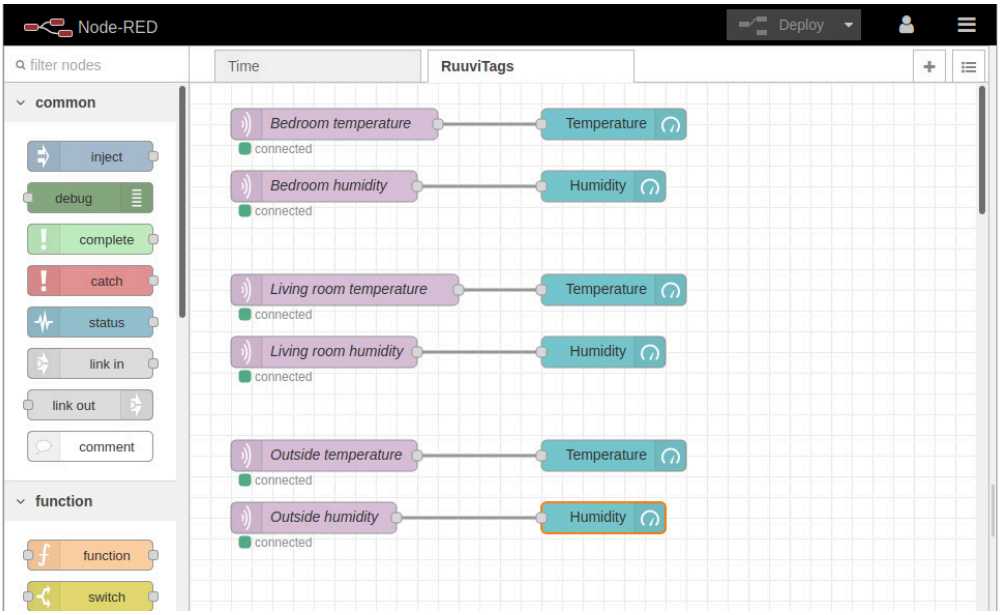

Figure 10.10 This flow lets each sensor measurement coming in through MQTT messages 'flow' to a dashboard widget.

If you have edited all the gauge nodes, click on **Deploy** to deploy your flows. If you visit https://HOSTNAME:1880/ui/ now, with the hostname of your Raspberry Pi, you first have to enter the username and password you configured earlier in this chapter for the dashboard, and afterwards you see your dashboard with some gauges that are updated continuously as the relevant MQTT messages are received by Node-RED.

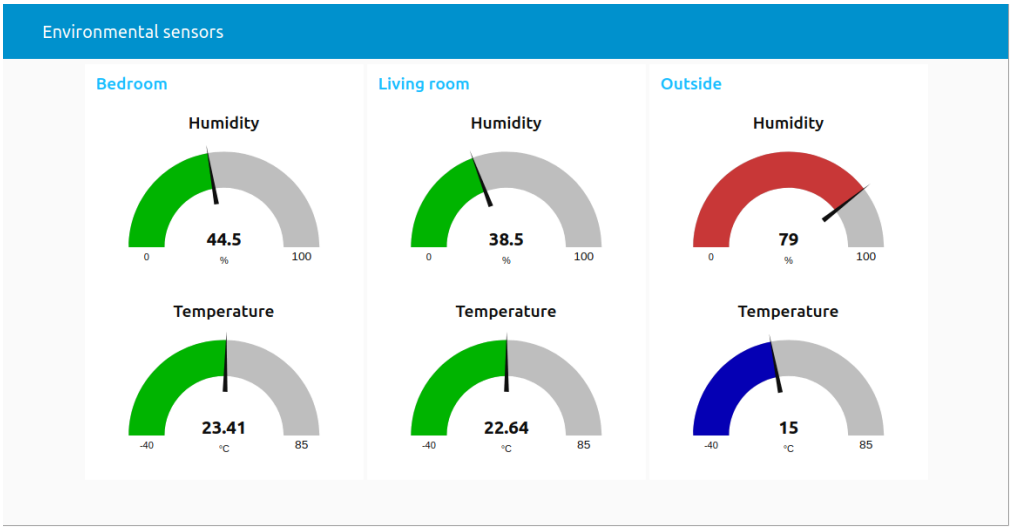

Figure 10.11 This simple Node-RED dashboard shows the sensor measurements from your environmental sensors.

Of course, this just scratched the surface of Node-RED's dashboard. Have a look at the other node types in the dashboard part of the palette and try them to create your own dashboards for your home automation system.

10.2 ● Home Assistant

Home Assistant (https://www.home-assistant.io) is a popular home automation gateway that puts local control and privacy first. It integrates with a lot of home automation devices and protocols, and it has a user-friendly way to add automations. It doesn't require you to write code, but has another approach than Node-RED's flow-based automations; instead, it uses automations with triggers, conditions, and actions.

10.2.1 ● Installing Home Assistant

To run Home Assistant as a Docker container, add this to your Docker Compose file:

```
version: '3.7'
services:
  mosquitto:
    # mosquitto configuration
  homeassistant:
    container_name: homeassistant
    image: homeassistant/home-assistant:latest
    volumes:
      - ./containers/homeassistant:/config
```

```
      - ./containers/certificates:/etc/ssl/private:ro
      - /etc/localtime:/etc/localtime:ro
    restart: always
    ports:
      - "8123:8123"
```

And create a directory where Home Assistant can store its data:

```
mkdir /home/pi/containers/homeassistant
```

Then start Home Assistant:

```
docker-compose up -d
```

After this, your Home Assistant setup is available on http://IP:8123, with IP the IP address of your Raspberry Pi. But let's immediately configure it to use TLS. Open the configuration file that Home Assistant has created:

```
sudo nano /home/pi/containers/homeassistant/configuration.yaml
```

Then add the following section:

```
http:
  base_url: https://HOSTNAME:8123
  ssl_certificate: /etc/ssl/private/cert.pem
  ssl_key: /etc/ssl/private/key.pem
```

Then restart the Home Assistant container:

```
docker-compose restart homeassistant
```

If you now visit https://HOSTNAME:8123 (make sure to use the hostname and not the IP address), your connection to Home Assistant is secured over TLS. Then follow the instructions of Home Assistant's configuration wizard: choose a username and password, and locate your home. When the wizard is ready, you see Home Assistant's web interface with the weather forecast for your location.

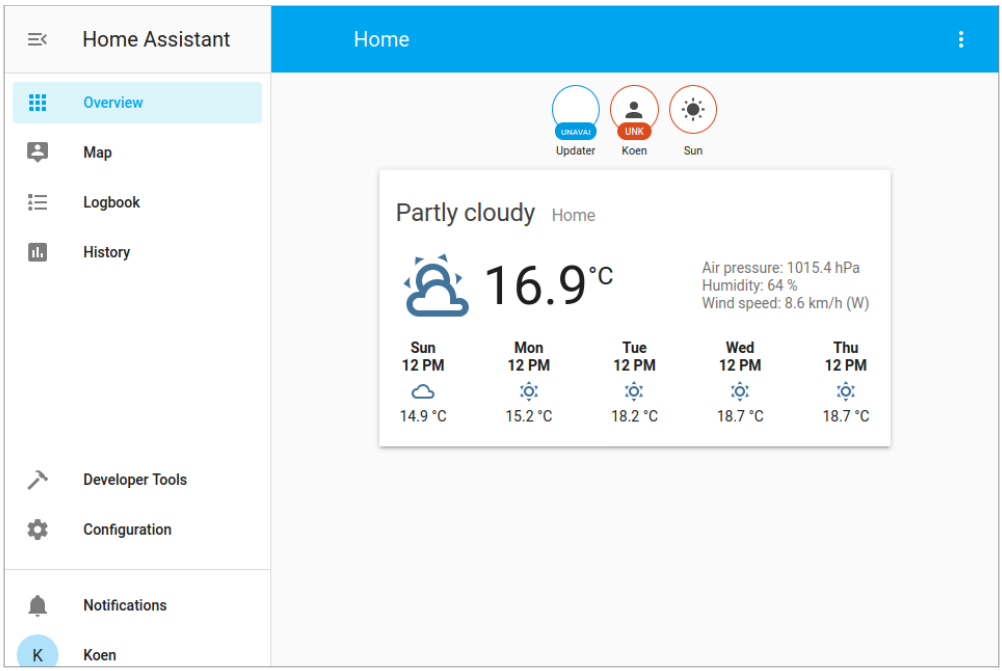

Figure 10.12 Home Assistant's configuration wizard makes the home automation software ready to use.

10.2.2 ● Integrating MQTT

Home Assistant supports integrations with a lot of home automation devices and services, but that's out of scope for this book. What's interesting for us is its MQTT integration. You can activate it by opening the menu **Configuration > Integrations**, clicking on the orange plus icon at the right bottom, and choosing **MQTT** in the list of integrations.

Then enter the connection for your MQTT broker. If you're using the Mosquitto container from the same Docker Compose file as the one where you defined the Home Assistant container, you can just enter **mosquitto** as the broker.[4] Enter your username and password. And then enable the **Enable discovery** checkbox for better integration with MQTT services that follow Home Assistant's discovery conventions. Click on **Submit** to add this configuration.

4 If you want to configure a connection to an MQTT broker with TLS, at the moment Home Assistant doesn't support this in a graphical way. Have a look at https://www.home-assistant.io/docs/mqtt/broker/ to see how to configure an encrypted MQTT broker connection in Home Assistant's configuration.yaml.

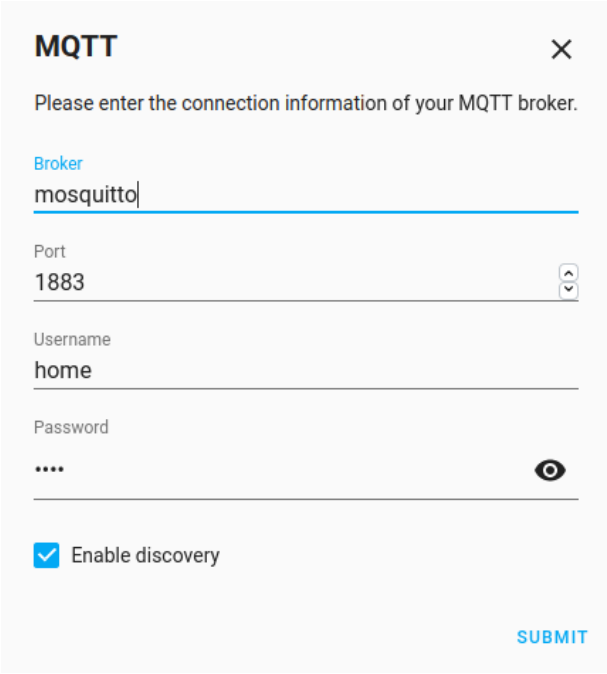

Figure 10.13 You can easily integrate Home Assistant with your MQTT broker.

Now Home Assistant will listen to messages published on the MQTT broker. It can even automatically discover devices thanks to the MQTT discovery conventions (https://www.home-assistant.io/docs/mqtt/discovery/).

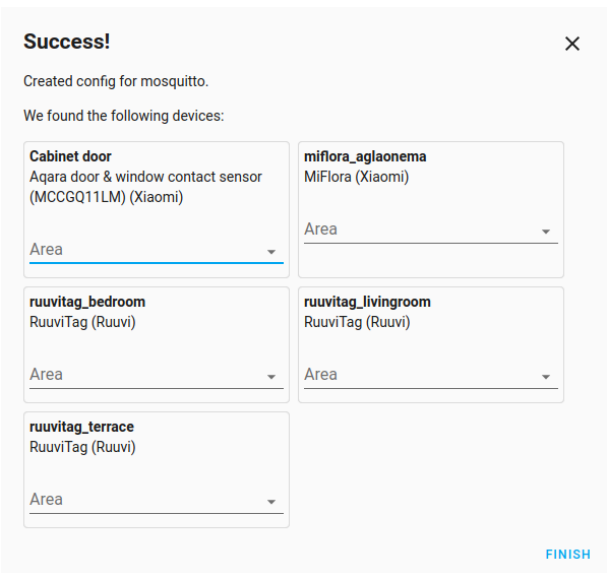

Figure 10.14 Home Assistant automatically adds devices that follow its MQTT discovery conventions.

Zigbee2mqtt has enabled MQTT discovery by default in its configuration file, so your Zigbee devices should appear automatically in Home Assistant. If not, make sure that Zigbee2mqtt's `configuration.yaml` has a line `homeassistant: true` and restart the Zigbee2mqtt container.

If you're using bt-mqtt-gateway (see Chapter 6), you only have to add some YAML code under the `manager` key in the project's configuration file:

```
mqtt:
  # mqtt config

manager:
  command_timeout: 30
  sensor_config:
    topic: homeassistant
    retain: true
  topic_subscription:
    update_all:
      topic: homeassistant/status
      payload: online
  workers:
    # worker configuration
```

After this, restart the bt-mqtt-gateway container:

```
docker-compose restart bt-mqtt-gateway
```

And if all goes well, you'll see the sensors you have defined in bt-mqtt-gateway appearing in Home Assistant's default view.

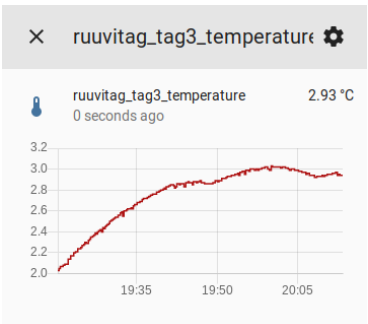

Figure 10.15 Thanks to MQTT discovery Home Assistant automatically adds sensors published by bt-mqtt-gateway.

10.2.3 ● Creating automation rules

Home Assistant also lets you create simple automation rules. Open **Configuration > Automations** and then click on the orange plus icon at the bottom right.

Home Assistant asks you to describe your automation rule and tries to convert this sentence (if it's in English) to an automation. In many cases this doesn't work yet, so just click on **Skip**. Give your automation rule a name and an optional description.

An Home Assistant automation rule has three parts:

Trigger
Triggers are what starts the processing of an automation rule. You can specify multiple triggers for the same rule. Once a trigger starts, Home Assistant validates the conditions, if any, and calls the action.

Condition
If a rule has a condition, the action is only called when the condition is true. So conditions can be used to prevent an action from happening when triggered in specific circumstances. Conditions are optional.

Action
Actions are what Home Assistant will do when the automation rule is triggered.

At first triggers and conditions look very similar, so the difference can be confusing. It helps to look at it like this:

- A trigger looks at events at the moment they are happening in the system. For instance, a trigger detects that a switch is being turned on.
- A condition looks at the current state of the system. For instance, a condition can check at any time if a switch is on or off.

To make it a bit more concrete: if you want to get a notification when the RuuviTag in your fridge reports a temperature higher than 5 degrees Celsius, you don't need a condition, you need a trigger.

Because temperature is a number, select **Numeric state** for the **Trigger type**. Then for **Entity**, choose your sensor's entity, such as `sensor.ruuvitag_tag1_temperature`. And enter **5** in the **Above** text field.

Then choose **Call service** for the **Action type** and `persistent_notification.create` for the **Service**. In the **Service data** text field, enter the following JSON code:

```
{
  "notification_id": "1234",
  "title": "Temperature alert",
  "message": "The fridge is too warm"
}
```

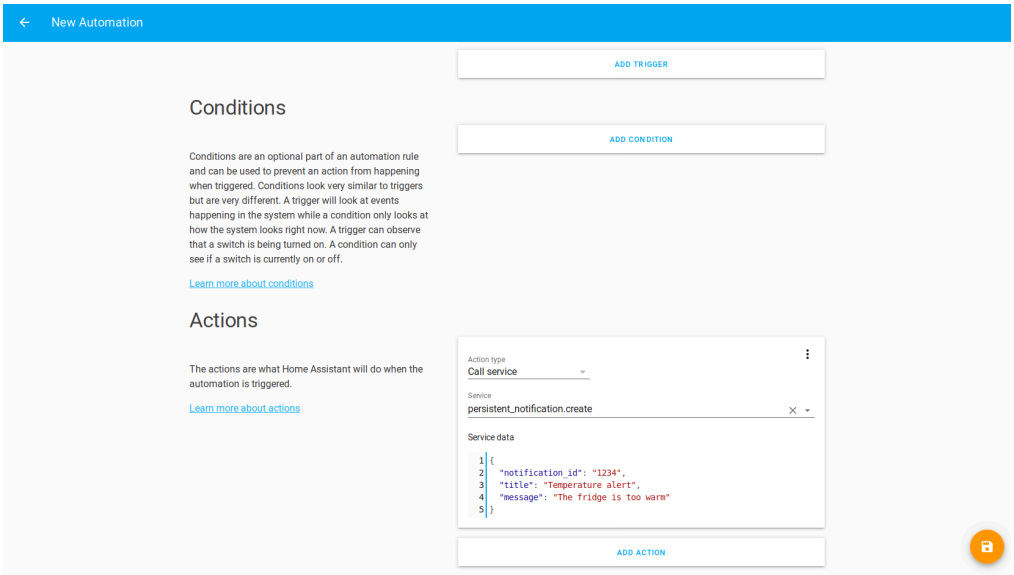

Figure 10.16 Home Assistant's automation rules have a simple trigger-condition-action structure.

Finally, click on the save icon to create your automation rule. If all goes well, you get a notification on Home Assistant's notification page (click on **Notifications** at the bottom left) when the temperature in your fridge is higher than 5 degrees Celsius.

Note:
This is just a simple example of a notification. Home Assistant knows a lot of more advanced notification types, including push notifications on mobile devices, text-to-speech notifications, SMS messages, emails, and messages on social networks. Have a look at https://www.home-assistant.io/integrations/#notifications for the complete list, and return to this chapter after reading the next chapter about notifications.

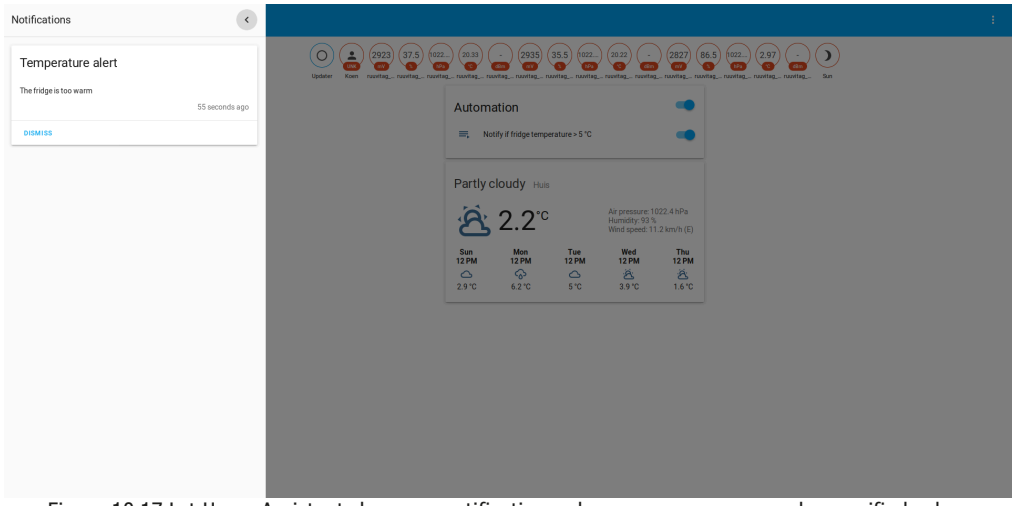

Figure 10.17 Let Home Assistant show you notifications when your sensors exceed a specified value.

10.3 ● AppDaemon

AppDaemon (https://appdaemon.readthedocs.io) is a loosely coupled, multithreaded Python execution environment for writing applications for home automation projects or any environment that requires a robust event-driven architecture.

AppDaemon is closely related to Home Assistant, so it has excellent support for Home Assistant, but it also directly supports MQTT. This makes it an interesting environment for Python programs that work with your home automation system's MQTT infrastructure. In this section, I'll show some basic examples.

10.3.1 ● Installing AppDaemon

AppDaemon has an official Docker image, but unfortunately, this isn't offered for the Raspberry Pi's ARM architecture.[5]

The solution is to build your own Docker image. So download the latest source and build its Docker image on your Raspberry Pi:[6]

5 You should check whether this still is true when you read this: https://appdaemon.readthedocs.io/en/latest/INSTALL.html#raspberry-pi-docker. I opened an issue about this in AppDaemon's GitHub repository: https://github.com/home-assistant/appdaemon/issues/970. If there will be an official ARM image, you should see this mentioned there.

6 Don't forget to regularly update AppDaemon's source with a `git pull` command in the appdaemon directory and then rebuild the Docker container with `docker build -t appdaemon`. After this, stop and remove the container with `docker stop appdaemon` and `docker rm appdaemon` and recreate it with `docker-compose up -d`.

```
git clone https://github.com/home-assistant/appdaemon.git
cd appdaemon
docker build -t appdaemon .
```

Building the Docker image takes a while (more than 20 minutes on a Raspberry Pi 4).

```
datalake-store, azure-keyvault, azure-servicemanagement-legacy, azure, pygments, sockjs, appdaemon
Successfully installed Jinja2-2.11.1 MarkupSafe-1.1.1 PyJWT-1.7.1 adal-1.2.2 aiohttp-3.6.2 aiohttp-jinja2
-1.2.0 appdaemon-4.0.4 astral-1.10.1 async-timeout-3.0.1 attrs-19.3.0 azure-4.0.0 azure-applicationinsigh
ts-0.1.0 azure-batch-4.1.3 azure-common-1.1.25 azure-cosmosdb-nspkg-2.0.2 azure-cosmosdb-table-1.0.6 azur
e-datalake-store-0.0.48 azure-eventgrid-1.3.0 azure-graphrbac-0.40.0 azure-keyvault-1.1.0 azure-loganalyt
ics-0.1.0 azure-mgmt-4.0.0 azure-mgmt-advisor-1.0.1 azure-mgmt-applicationinsights-0.1.1 azure-mgmt-autho
rization-0.50.0 azure-mgmt-batch-5.0.1 azure-mgmt-batchai-2.0.0 azure-mgmt-billing-0.2.0 azure-mgmt-cdn-3
.1.0 azure-mgmt-cognitiveservices-3.0.0 azure-mgmt-commerce-1.0.1 azure-mgmt-compute-4.6.2 azure-mgmt-con
sumption-2.0.0 azure-mgmt-containerinstance-1.5.0 azure-mgmt-containerregistry-2.8.0 azure-mgmt-container
service-4.4.0 azure-mgmt-cosmosdb-0.4.1 azure-mgmt-datafactory-0.6.0 azure-mgmt-datalake-analytics-0.6.0
azure-mgmt-datalake-nspkg-3.0.1 azure-mgmt-datalake-store-0.5.0 azure-mgmt-datamigration-1.0.0 azure-mgmt
-devspaces-0.1.0 azure-mgmt-devtestlabs-2.2.0 azure-mgmt-dns-2.1.0 azure-mgmt-eventgrid-1.0.0 azure-mgmt-
eventhub-2.6.0 azure-mgmt-hanaonazure-0.1.1 azure-mgmt-iotcentral-0.1.0 azure-mgmt-iothub-0.5.0 azure-mgm
t-iothubprovisioningservices-0.2.0 azure-mgmt-keyvault-1.1.0 azure-mgmt-loganalytics-0.2.0 azure-mgmt-log
ic-3.0.0 azure-mgmt-machinelearningcompute-0.4.1 azure-mgmt-managementgroups-0.1.0 azure-mgmt-managementp
artner-0.1.1 azure-mgmt-maps-0.1.0 azure-mgmt-marketplaceordering-0.1.0 azure-mgmt-media-1.0.0 azure-mgmt
-monitor-0.5.2 azure-mgmt-msi-0.2.0 azure-mgmt-network-2.7.0 azure-mgmt-notificationhubs-2.1.0 azure-mgmt
-nspkg-3.0.2 azure-mgmt-policyinsights-0.1.0 azure-mgmt-powerbiembedded-2.0.0 azure-mgmt-rdbms-1.9.0 azur
e-mgmt-recoveryservices-0.3.0 azure-mgmt-recoveryservicesbackup-0.3.0 azure-mgmt-redis-5.0.0 azure-mgmt-r
elay-0.1.0 azure-mgmt-reservations-0.2.1 azure-mgmt-resource-2.2.0 azure-mgmt-scheduler-2.0.0 azure-mgmt-
search-2.1.0 azure-mgmt-servicebus-0.5.3 azure-mgmt-servicefabric-0.2.0 azure-mgmt-signalr-0.1.1 azure-mg
mt-sql-0.9.1 azure-mgmt-storage-2.0.0 azure-mgmt-subscription-0.2.0 azure-mgmt-trafficmanager-0.50.0 azur
e-mgmt-web-0.35.0 azure-nspkg-3.0.2 azure-servicebus-0.21.1 azure-servicefabric-6.3.0.0 azure-servicemana
gement-legacy-0.20.6 azure-storage-blob-1.5.0 azure-storage-common-1.4.2 azure-storage-file-1.4.0 azure-s
torage-queue-1.4.0 bcrypt-3.1.7 certifi-2020.4.5.1 cffi-1.14.0 chardet-3.0.4 cryptography-2.9 deepdiff-4.
3.1 feedparser-5.2.1 idna-2.9 iso8601-0.1.12 isodate-0.6.0 msrest-0.6.13 msrestazure-0.6.3 multidict-4.7.
5 oauthlib-3.1.0 ordered-set-3.1.1 paho-mqtt-1.5.0 pid-2.2.5 pycparser-2.20 pygments-2.6.1 python-dateuti
l-2.8.1 python-engineio-3.12.1 python-socketio-4.4.0 pytz-2019.3 pyyaml-5.3 requests-2.23.0 requests-oaut
hlib-1.3.0 six-1.14.0 sockjs-0.10.0 urllib3-1.25.9 voluptuous-0.11.7 websocket-client-0.57.0 yarl-1.4.2
Removing intermediate container b6c6dc6e376c
 ---> 7065653cd68b
Step 9/11 : RUN apk add --no-cache curl
 ---> Running in efa9e2e17d91
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/main/armv7/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/community/armv7/APKINDEX.tar.gz
(1/3) Installing nghttp2-libs (1.40.0-r0)
(2/3) Installing libcurl (7.67.0-r0)
(3/3) Installing curl (7.67.0-r0)
Executing busybox-1.31.1-r9.trigger
OK: 81 MiB in 53 packages
Removing intermediate container efa9e2e17d91
 ---> de8303bf8095
Step 10/11 : RUN chmod +x /usr/src/app/dockerStart.sh
 ---> Running in fb542ac46614
Removing intermediate container fb542ac46614
 ---> d702fb2dc05b
Step 11/11 : ENTRYPOINT ["./dockerStart.sh"]
 ---> Running in fe8c1e4d27d3
Removing intermediate container fe8c1e4d27d3
 ---> ce1a942c660a
Successfully built ce1a942c660a
Successfully tagged appdaemon:latest
pi@pi-red:~/appdaemon $
```

Figure 10.18 You have to build the AppDaemon Docker image for the Raspberry Pi yourself.

When the image has been built, you can use it in your Docker Compose file:

```
version: '3.7'

services:
  mosquitto:
    # mosquitto configuration
  homeassistant:
    # homeassistant configuration
  appdaemon:
    image: appdaemon
    container_name: appdaemon
    restart: always
    volumes:
      - ./containers/appdaemon:/conf
    user: "1000:1000"
```

Note that you use just `appdaemon` as the image name, and not `acockburn/appdaemon` (which is the official Docker image of AppDaemon). By using the `appdaemon` tag you refer to the image you built yourself: you supplied this tag with the `-t appdaemon` option in the `docker build` command).

Then create the directory for AppDaemon:

```
mkdir /home/pi/containers/appdaemon
```

And create a file `appdaemon.yaml` in this directory. A minimal configuration to use AppDaemon with MQTT should look like this:

```
appdaemon:
  time_zone: Europe/Brussels
  latitude: LATITUDE
  longitude: LONGITUDE
  elevation: ELEVATION
  plugins:
    MQTT:
      type: mqtt
      client_host: mosquitto
      namespace: mqtt
      client_user: home
      client_password: PASSWORD
```

In this file, you have to enter the timezone, latitude, longitude, and elevation for your

location.[7]

After this comes the configuration of the MQTT plugin. The value of `client_host` should be the hostname or IP address of the MQTT broker. Because you're running Mosquitto as a Docker container configured in the same Docker Compose file as AppDaemon, you can use `mosquitto` here, as it's the name of the Mosquitto container.[8]

After the configuration is done, start the container:

```
docker-compose up -d
```

This will create an example app. You can delete it because you don't need it:

```
rm -rf /home/pi/appdaemon/apps/*
```

10.3.2 ● Creating an AppDaemon app with MQTT: the time

Now revisit the Python program in Chapter 4 that sends an MQTT message with the time to your broker when someone asks for it. I'll rewrite this to an AppDaemon app. Create a directory for the app:

```
mkdir /home/pi/containers/appdaemon/apps/time_app
```

Then the app's code looks like this:

```
"""Send an MQTT message with the time to your broker when asked.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
import mqttapi as mqtt
```

7 It may look strange that these parameters are mandatory, especially the latitude, longitude and elevation, but they are used by AppDaemon to run functions at sunrise or sunset.

8 If you want to use an external MQTT broker with TLS, consult AppDaemon's documentation for the configuration of the MQTT plugin (https://appdaemon.readthedocs.io/en/latest/CONFIGURE.html#configuration-of-the-mqtt-plugin).

```
MQTT_MSG = "MQTT_MESSAGE"
MQTT_TOPIC_TIME_REQUEST = "time/request"
MQTT_TOPIC_TIME_REPLY = "time/reply"
TIME_FORMAT = "%Y-%m-%d %H:%M"


class TimeApp(mqtt.Mqtt):
    """App that sends the time as an MQTT message."""

    def initialize(self):
        """Subscribe to the right MQTT topic."""
        self.set_namespace("mqtt")
        self.listen_event(
         self.on_time_request, event=MQTT_MSG, topic=MQTT_TOPIC_TIME_REQUEST
        )
        self.log("Time app initialized")

    def on_time_request(self, event_name, data, kwargs):
        """Reply with the time if it's asked."""
        now = self.datetime().strftime(TIME_FORMAT)
        self.mqtt_publish(MQTT_TOPIC_TIME_REPLY, now)
```

It's immediately clear that you need much less boilerplate code. You don't need to define the MQTT host and port, because it's already defined in AppDaemon's configuration. You also don't need to initialize the MQTT connection and start the event loop. So the only thing you need to do is start listening to the right MQTT topic, with the `self.listen_event` line.

Note:
Instead of `datetime.now()` from Python's standard library, this AppDaemon app uses `self.datetime()` to get the current date and time. This is the preferred way in AppDaemon.

Now if you change your MQTT broker, the only thing you have to change is the configuration of AppDaemon. After restarting AppDaemon, all AppDaemon apps use this new MQTT broker. Compare this to the approach in Chapter 4: you had to change the MQTT settings in all Python files.

Save this file as `time_app.py` in the `apps/time_app` directory of AppDaemon, and also create a file called `time_app.yaml` in the same directory. Give this file the following content:

```
time_app:
  module: time_app
  class: TimeApp
```

After you have saved both files, AppDaemon automatically picks up the changes and loads your app. You don't have to 'run' the app or restart AppDaemon for this. So now you can just test the app by subscribing to the `time/reply` topic in one shell:

```
mosquitto_sub -t 'time/reply' -v
```

In another shell, send a request to the program

```
mosquitto_pub -t 'time/request' -m ''
```

If all goes well, you'll see the current date and time appearing in the shell running `mosquitto_sub`.

10.3.3 ● Creating an AppDaemon app with MQTT: garage door alert

Now let's revisit the Python program from Chapter 6 that uses a RuuviTag attached to your garage door to see whether it's open or closed. Create a directory for the app:

```
mkdir /home/pi/containers/appdaemon/apps/garage_door
```

If you use bt-mqtt-gateway to translate the BLE data from the RuuviTag to MQTT messages, an AppDaemon app would look like this:

```
"""Send an MQTT message to your broker when the position of your
garage door changes.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
import mqttapi as mqtt

MQTT_MSG = "MQTT_MESSAGE"
MQTT_TOPIC_ACCEL = "bt-mqtt-gateway/ruuvitag/tag2/acceleration_z"
MQTT_TOPIC_STATE = "garagedoor/state"


class GarageDoor(mqtt.Mqtt):
    """App that signals the position of your garage on MQTT."""
```

```
    def initialize(self):
        """Subscribe to the right MQTT topic."""
        self.set_namespace("mqtt")
        self.state = "error"
        self.listen_event(self.on_accel, event=MQTT_MSG, topic=MQTT_TOPIC_
ACCEL)
        self.log("Garage door app initialized")

    def change_and_publish_if_not(self, state):
        """Change and publish the garage door's state
        if the current state is not equal to the state.
        Do nothing in the other situation.
        """
        if self.state != state:
            self.state = state
            self.mqtt_publish(MQTT_TOPIC_STATE, state)

    def on_accel(self, event_name, data, kwargs):
        """Publish the garage door's state when it changes."""
        acceleration_z = int(data["payload"])
        if acceleration_z > 100:
            self.log("Right side up")
            self.change_and_publish_if_not("error")
        elif acceleration_z < -100:
            self.log("Upside down")
            self.change_and_publish_if_not("open")
        else:
            self.log("On its side")
            self.change_and_publish_if_not("closed")
```

You'll have to change the `MQTT_TOPIC_ACCEL` topic to the one you have configured for your RuuviTag. After this, the app just listens to messages on this topic and emits the right MQTT message when the state of the Z component of the acceleration changes.

Save this file as `garage_door.py` in the `apps/garage_door` directory of AppDaemon, and also create a file called `garage_door.yaml` in the same directory. Give this file the following content:

```
garage_door:
  module: garage_door
  class: GarageDoor
```

Now when the RuuviTag on your garage door changes position, the state is sent to the garagedoor/state topic.

10.4 ● Summary and further exploration

This chapter was all about more maintainable ways to automate your home than a bunch of Python scripts. I introduced you to three interesting home automation platforms: Node-RED, Home Assistant, and AppDaemon. The first two are interesting because they have a complete ecosystem of integrations and add-ons around them, so you don't have to reinvent the wheel. AppDaemon is interesting if you want to automate your home with Python but do it in a more maintainable way.

There are more projects than these three. Many home automation platforms are popular in specific countries or language regions. For instance, in Germany FHEM (https://fhem.de) is a popular choice, in France Jeedom (https://jeedom.com) and in the Netherlands Domoticz (https://www.domoticz.com). Other popular open-source home automation platforms are openHAB (https://www.openhab.org) and ioBroker (https://www.iobroker.net).

A fairly recent addition is Mozilla IoT WebThings Gateway (https://iot.mozilla.org/gateway). This innovative project is trying to create a decentralized Internet of Things by giving Things URLs on the web to make them linkable and discoverable. This "Web of Things" is a great vision, but it's not yet as mature and usable as the other projects in this chapter. An interesting project that builds on the IoT WebThings Gateway is Candle (https://www.candlesmarthome.com), a prototype of a privacy-friendly smart home solution developed by a collective of designers, artists, and privacy experts from Amsterdam.

If you want to use one of these instead of the solutions in this chapter, feel free to do this. They all have an open architecture and all run on a Raspberry Pi, so you should be able to use them with the approach advocated in this book.

Chapter 11 • Notifications

Your home automation gateway can do a lot of things automatically, and it can show you some information on a dashboard. But sometimes that's not enough: you want to be notified about some events. In this chapter, I'll show you two ways to send notifications: emails and push notifications with Gotify.

These two ways cover most use cases. Emails are useful to send you information that you don't have to act upon immediately, and push notifications can give you immediate warnings when that's needed.

I end this chapter with mqttwarn, a highly flexible system that you can let react to specific MQTT topics and then notify you using a wide range of notification services. This is an easy way to send notifications without having to program.

Note:
Setting up a fully self-hosted notifications infrastructure is no easy feat. Email on the server of your email provider is not self-hosted. However, I put email in this chapter because you are free to choose your email provider. For push notifications, there's much less choice. If you choose a centralized notifications service such as Pushover, you depend on one company, which is what I'm trying to avoid in this book (see Chapter 1). That's why I use Gotify, which you can install on your server.

11.1 • Forwarding local email

Email is probably the simplest way to send you notifications. It's not very useful if you want to be notified immediately, but for a lot of events, it's the perfect type of notification. For instance, if you have set up Raspberry Pi OS to automatically update all packages daily (see Chapter 3), you don't need to see a notification immediately: it's enough if you read a report of the updated packages when you open your inbox.

Many of the system services on Raspberry Pi OS can send you emails, for example, if someone tries to log in with a wrong password. By default, these emails arrive in a local mailbox on your Raspberry Pi, where you probably won't read these. Therefore it's much more useful to forward these locally delivered emails to your normal email address.

You can do this by installing a full-blown mail server on your Raspberry Pi, such as Postfix or Exim, but that's probably overkilling. A better alternative is Nullmailer, which is specifically developed for this purpose: forwarding locally delivered emails to a configured email address.

11.1.1 • Installing Nullmailer

Installing Nullmailer is simple:

```
sudo apt install nullmailer
```

This will ask you to confirm or change some settings. Just accept the default settings; you'll change them afterwards in the configuration files in the `/etc/nullmailer` directory.

Note:
You could also install nullmailer in a Docker container, but I prefer to run nullmailer on Raspberry Pi OS directly, so it still works when your Docker setup breaks.

The most important configuration is the mail server to deliver emails to. Open the configuration file:

```
sudo nano /etc/nullmailer/remotes
```

And then enter a line with your mail server's configuration. This should look like this:

```
smtp.example.com smtp --user=USERNAME --pass=PASSWORD --port=587 --starttls
```

Let's look at each of these components:

`smtp.example.com`
This is the hostname of the SMTP (Simple Mail Transfer Protocol) server of your mail server.

`--user=USERNAME`
Replace `USERNAME` by your username on the mail server. For some mail servers, this username is only the part before the @ sign in your email address; for others, this is your full email address.

`--pass=PASSWORD`
Replace `PASSWORD` by your email password. Note: if your password contains spaces, add it as `--pass='my password'`.

`--port=587`
This is the port number where the SMTP server is listening to.

`--starttls`
This option initiates a TLS session with the STARTTLS command.
You should consult your email provider's documentation for some of these values. Look up other options in `man nullmailer-send` or `/usr/lib/nullmailer/smtp --help`.

Warning:
The /etc/nullmailer/remotes file contains your email password in plaintext. This makes it especially important to secure your Raspberry Pi (see Chapter 3) so no one can find your email password. To be on the safe side, you should use a dedicated email account for your Raspberry Pi, so if someone finds your email password in this file, your main email account isn't compromised.

Now you have to configure the sender of the emails sent by Nullmailer. This is done in another configuration file:

```
sudo nano /etc/nullmailer/allmailfrom
```

You should enter a valid email address in this file, preferably the email address from the account that you configured in the `/etc/nullmailer/remotes` file. Otherwise, your mail server will probably reject your emails.

The next step is to configure Nullmailer to forward all locally delivered emails to your email address. Enter this email address in another file:

```
sudo nano /etc/nullmailer/adminaddr
```

11.1.2 ● Testing Nullmailer

After this, you should test your configuration. Stop the `nullmailer` service and start `nullmailer` manually to look at its output:

```
sudo systemctl stop nullmailer
sudo nullmailer-send
```

Now locally delivered emails should be forwarded to your configured email address. You can test this manually. First, install the `mail` command with:

```
sudo apt install mailutils
```

After this, send a test email with:

```
echo "This is a mail message" | mail -s "This is a mail subject" root@localhost
```

If all goes well, the running `nullmailer-send` command should show the message **Delivery complete, 0 message(s) remain**., and your test email arrives in your mailbox. If not, try to understand what the error message in the output of `nullmailer-send` is telling you. In this case, you probably also receive an email from your mail server with an error message with more information.

11.1.3 ● Using Nullmailer

If sending an email works, you can exit the `nullmailer-send` command with Ctrl+C and then restart the `nullmailer` service:

```
sudo systemctl start nullmailer
```

All the system services on your Raspberry Pi that you configure to send email to local users now work using Nullmailer and forward those emails to your email address with your email provider

11.2 ● Forwarding emails from Docker containers

If you want services in your Docker containers, such as motionEye (for motion detection, see Chapter 5), Node-RED, or Home Assistant (see Chapter 10) to send emails to you as notifications, the previous setup doesn't suffice. But the installation and configuration of a full-blown mail server for your local network is beyond the scope of this book.
However, there's an easier solution: an SMTP relay that accepts emails from other containers on the same Raspberry Pi and sends them to an external mail server. One simple project that does this, is the docker-postfix project (https://github.com/juanluisbaptiste/docker-postfix) by Juan Luis Baptiste.[1]

11.2.1 ● Installing docker-postfix

Unfortunately, at the moment, the docker-postfix project only publishes container images for the amd64 architecture on Docker Hub, so you can't run them on a Raspberry Pi's ARM processor.[2] So you have to build the image yourself:

1 Postfix (http://www.postfix.org) is an easy to administer and secure mail transfer agent (MTA).
2 See https://github.com/juanluisbaptiste/docker-postfix/issues/23 for the status of the publication of ARM images of the project on Docker Hub.

```
git clone https://github.com/juanluisbaptiste/docker-postfix.git
cd docker-postfix
git checkout migrate_to_alpine
docker build -t postfix .
```

The third line is only needed to switch to the branch that uses Alpine Linux as the base operating system for the container image. If entering this line on the command-line results in an error, the Alpine Linux image has been made the default one and you can just skip this line.

After the Docker image has been built, you can use it in your `docker-compose.yml` file:

```
version: '3.7'

services:
  # other services
  postfix:
    image: postfix
    container_name: postfix
    restart: always
    environment:
      SMTP_SERVER: SMTP_SERVER
      SMTP_PORT: 587
      SMTP_USERNAME: USERNAME
      SMTP_PASSWORD: PASSWORD
      SERVER_HOSTNAME: HOSTNAME
    volumes:
      - /etc/localtime:/etc/localtime:ro
```

As the image, you just enter `postfix`, because it's your locally built postfix image. If by the time you read this Juan Luis has published ARM images on Docker Hub, you don't have to build it yourself anymore and can just refer to the `juanluisbaptiste/postfix` image in your Docker Compose file.

The configuration settings of the server you want to relay your emails to are set in a couple of environment variables.

Note:
There are no port mappings defined for the container in this Docker Compose file. This is how you make the mail server only accessible to the other Docker containers defined in the Docker Compose file.

After this, create the container with:

```
docker-compose up -d
```

Now look at the logs of the postfix container:

```
docker logs -f postfix
```

If all goes well, you should see a successful start of the Postfix container. If not, check the configuration of your mail server settings.

11.2.2 ● Sending emails to docker-postfix

Now your other containers defined in the Docker Compose file on the same Raspberry Pi can send emails to your docker-postfix container on port 25. I give an example of how you do this in Node-RED (see Chapter 10).

In Node-RED, install the `node-red-node-email` module. Two new nodes are added in the **social** category: **email in** (for receiving emails) and **email** (for sending emails). Drag the second on the canvas and add a **change** node before it, and an **inject** node before the **change** node. Connect the three nodes so the flow goes from **inject** to **change** to **email**.

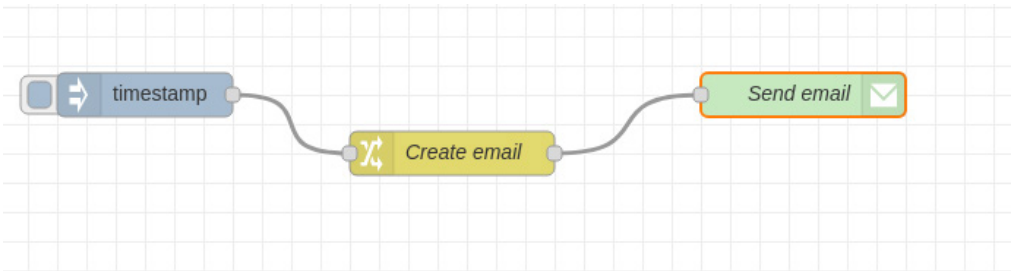

Figure 11.1 A simple flow to test sending emails from Node-RED to docker-postfix.

Now double-click on the email node, enter the email address of the receiver, enter **postfix** (the name of the container running the mail server) as the server address, and **25** as the port number. Uncheck **Use secure connection** and **Use TLS?**. Leave the user ID and password empty. Click on **Done** to save the node.

Warning:
As these settings make clear, the docker-postfix mail server doesn't require any form of authentication or encryption. That's why it's important to not expose the container's port 25 to the Raspberry Pi itself because then anyone on your network can send emails using your mail server. This set up is only meant for allowing Docker containers on the same machine send emails outside your network.

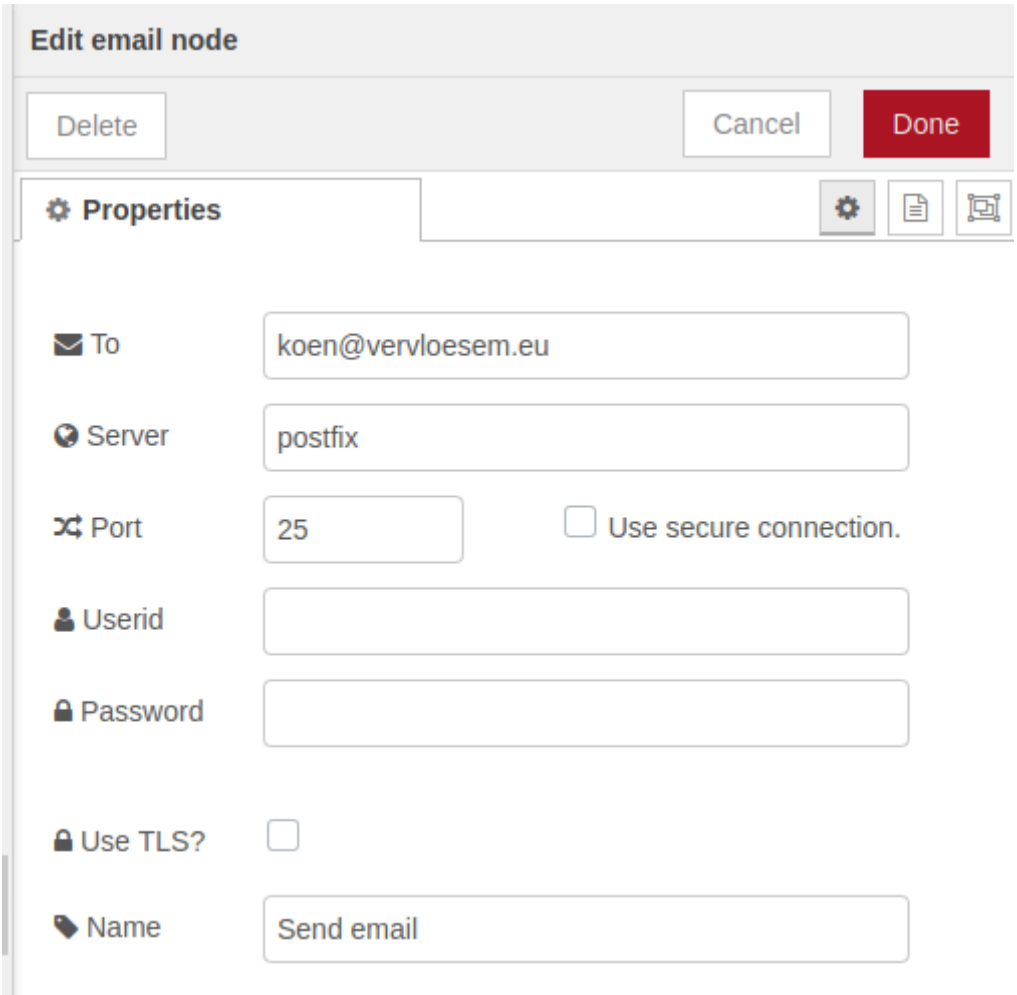

Figure 11.2 Enter the settings for your docker-postfix mail server.

Then double-click on the change node set **msg.from** to the email address of the sender (many mail servers require a valid sender address), **msg.payload** to the body of the email, and **msg.topic** to the title of the email. Save your settings, deploy the flow, and then click on the button at the left of the inject node. If all goes well, you're receiving an email from Node-RED now. If not, look at the logs of docker-postfix: the mail server has probably

rejected the email and lists the reason.

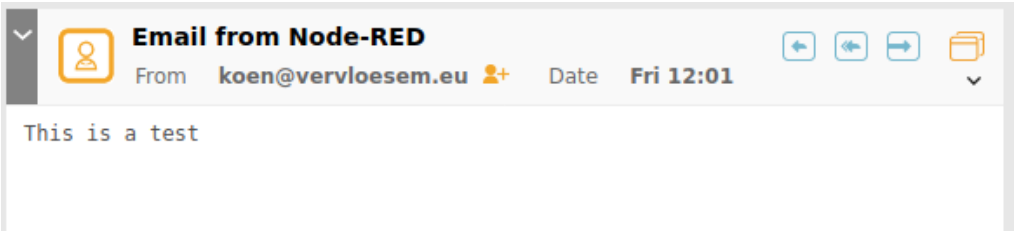

Figure 11.3 You've got mail from Node-RED.

If this test works, you can now send emails in more complex Node-RED flows to notify you about important events. The process is similar for other containers: just enter postfix as the server and 25 as the port number.

11.3 ● Push notifications with Gotify

Email is a nice way for notifications, but if you want to be notified immediately when something happens, push notifications are better. We all know push notifications from our smartphones. Many of these systems work with a central server where all notifications are sent. This central server then sends the notifications to the right device or app.

Almost always, this central server is managed by a commercial entity and is not under your control. This means that you have to trust a third party that processes your notifications. It also means that you probably have to pay for the service or be happy with some restrictions on the number of notifications you can send or other aspects of the service.
However, there's an alternative that is open-source and that you can run on your system, such as on the Raspberry Pi running your home automation system: Gotify (https://gotify.net). It has a simple to use web interface, can manage users, clients and applications and lets you send messages via REST and subscribe to and receive messages via a WebSocket connection.

The only downside is that Gotify doesn't support iOS: for smartphones, there's only an Android app. The problem seems to be that Apple has strict restrictions on background services, so an iOS app for Gotify wouldn't be able to keep a persistent WebSocket connection in the background.[3] The only way to send notifications on iOS devices is using the Apple Push Notification service, which works with a central server to manage notifications and sends them to Apple before reaching the user. This is not what Gotify is designed for, nor does it fit in this book's approach.

3 See https://github.com/gotify/server/issues/87

Note:
If you set up your notification server on your LAN, Gotify will only be able to notify your phone as long as it's connected to the LAN, not when you're away from home. However, notifications will work when you're away frome home and connected to your VPN. See Chapter 13 to set up a VPN for remote access. You could also set up Gotify on a VPS (virtual private server) on the internet.

There are three parts in Gotify's architecture:

The Gotify server
This is the server with a REST and WebSocket API that receives notifications from applications and delivers them to clients.

Gotify applications
Programs that send messages to the Gotify server using the REST API.

Gotify clients
Programs that receive notifications from the Gotify server using the WebSocket API. Examples of clients are the web interface of the server and an Android app.

11.3.1 ● Installing the Gotify server

Gotify can be installed easily as a Docker container. Edit your `docker-compose.yml` file in your home directory to have the following content:

```
version: '3.7'

services:
  # Other services
  gotify:
    image: gotify/server-arm7
    container_name: gotify
    restart: always
    volumes:
      - ./containers/gotify:/app/data
      - ./containers/certificates:/certs:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - 8443:443
    environment:
      - GOTIFY_SERVER_SSL_ENABLED=true
      - GOTIFY_SERVER_SSL_CERTFILE=/certs/cert.pem
      - GOTIFY_SERVER_SSL_CERTKEY=/certs/key.pem
      - GOTIFY_DEFAULTUSER_PASS="!K8B*vuSixsb2tAn&BIA"
```

Change the values of the environment variables for your certificate and key file and choose a strong password for the default user (admin) in the `GOTIFY_DEFAULTUSER_PASS` variable.

Note:
If your password contains special characters, you have to quote it in the Docker Compose file, so "!K8B*vuSixsb2tAn&BIA" instead of !K8B*vuSixsb2tAn&BIA.

Now start the container:

```
docker-compose up -d
```

After this, you should be able to log into Gotify's web interface (on https://HOSTNAME:8443) with the username `admin` and the password you configured in the `docker-compose.yml` file.

You can now create users for everyone in your household (click on **Users** at the top), or you can just use the admin account if you're the only user.

11.3.2 ● Adding applications to Gotify

Now you should create an application in the web interface for every program that sends push notifications with Gotify. For instance, if you want Home Assistant to send push notifications, then create an application **Home Assistant**.

Click on **Apps** at the top and then on **Create application**. Provide a name and (optionally) a short description. Click on **Create**.

Every application that you create this way comes in a list, with a token and an icon. If you have multiple applications, it's recommended to upload a custom image (PNG, JPEG, or GIF) for each application with the upload icon right to the default icon. That way you easily recognize the source of notifications.

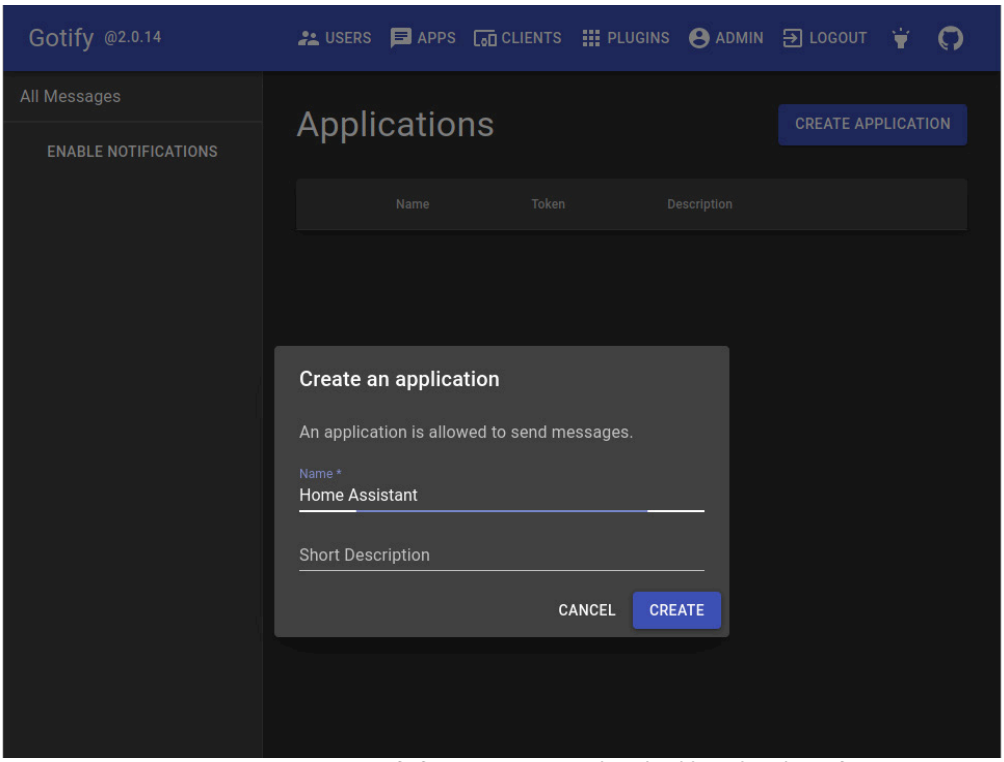

Figure 11.4 Create an app in Gotify for every program that should send push notifications.

For each application, a tab with its name is added to the left sidebar, under **All Messages**. If you click on it, this shows only the messages from this specific application.

11.3.3 ● Using Gotify applications

After you have added one or more applications to Gotify, you can start using them to send messages. How exactly this works depends on the application, but they all use REST (see also Chapter 5), so they all need the same information:

The REST URI
This has the form `https://HOSTNAME:8443/message?token=<apptoken>`, where `<apptoken>` is the token assigned to your application in the **Apps** list.

URI parameters
You choose the content of the notification by URI parameters `message`, `title`, and `priority`. Only the first one is required.

An example makes this clear. You can test push notifications with this one-liner on the command line:

```
curl "https://HOSTNAME:8443/message?token=APPTOKEN" --cacert /home/pi/⏎
containers/certificates/rootCA.pem -F "title=First" -F "message=My first ⏎
push notification" -F "priority=5"
```

Note the `--cacert` option, which specifies the root CA certificate used to verify the TLS certificate of your Gotify server. Without this option, curl will fail to connect.

After this, the curl command gives some output in JSON format. If you go to the web interface, you'll see the notification too, in the **All Messages** tab as well as the tab from your application.

Note:
If you don't like the verbosity of curl commands, there are some command-line alternatives. The developers of Gotify have Gotify-CLI (https://github.com/gotify/cli), which stores the REST URI and application token in a configuration file. After this, you can just enter gotify push "My message". Another command-line client is gotify-push (https://github.com/schwma/gotify-push), that supports multiple applications and users.

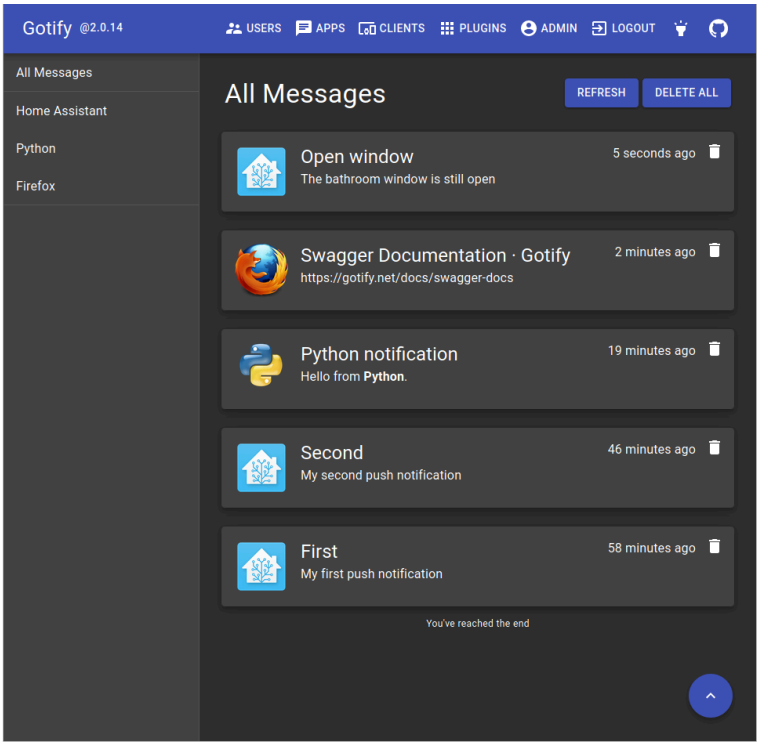

Figure 11.5 Gotify's web interface shows the messages sent by all your applications.

Now you can use the same approach to send notifications from:

Node-RED
with a http request node

Home Assistant
with the rest notification platform (https://www.home-assistant.io/integrations/notify.rest)

motionEye
by enabling Call A Web Hook in Motion Notifications

You can also send a notification in your Python programs. A simple example would look like this:

```
"""Post a notification to your Gotify server.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
import requests

resp = requests.post(
    "https://HOSTNAME:8443/message?token=APPTOKEN",
    json={
        "message": "Hello from **Python**.",
        "priority": 2,
        "title": "Python notification",
        "extras": {"client::display": {"contentType": "text/markdown"}},
    },
    verify="/home/pi/containers/certificates/rootCA.pem",
)
```

This is using the Requests library (https://requests.readthedocs.io), which makes it very easy to send HTTP requests (see also Chapter 5). You can install it with `pip3 install requests`. Change the hostname and app token to your situation and specify the location of your root CA certificate on the last line to verify the TLS certificate of the Gotify server.

Note that this code uses message extras (https://gotify.net/docs/msgextras#clientdisplay) to set the content type to Markdown. This way you can use formatting codes to spice up the layout of your notification messages, such as `**Python**` to make Python bold.

There are also some browser add-ons, such as Gotify for Firefox (https://addons.mozilla.org/nl/firefox/addon/gotify-for-firefox/) and Gotify Chrome (https://github.com/rorpage/

gotify-chrome) that let you send notifications to Gotify from your web browser. For instance, you can push the current page's URL or you can create a note and send it via the add-on's icon in the top right corner. After installing the add-on, you first have to enter the REST API URL of the Gotify server and the application token that you have created for the browser add-on.

11.3.4 ● Using Gotify clients

This is all nice, but looking at a web interface is not the same as getting realtime notifications on your phone. The project has an official Android app, called Gotify (https://play.google.com/store/apps/details?id=com.github.gotify). This app subscribes to events from your Gotify server's WebSocket API and creates push notifications on any new messages.

After you have installed the app on your Android phone, enter the URL of your Gotify server and press **Check URL**. Add your username (**admin** if you haven't created another user) and your password, and press **Login**. Then choose a name for your client and press **Create**.

After this setup, the app on your phone shows you the same list of notifications as the web interface. The hamburger menu at the left lets you filter all notifications from specific applications.

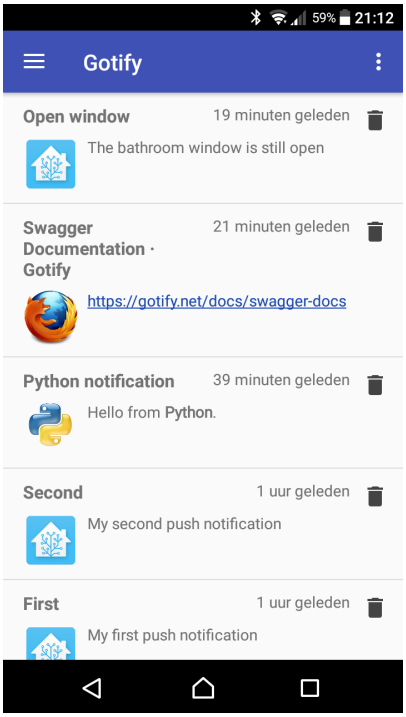

Figure 11.6 Your Gotify server pushes notifications to your Android phone.

But the whole point of this app is of course that you can receive push notifications. When one of your applications sends a message, you receive an immediate notification on your Android phone.

If you set up Gotify applications for the components in your home automation system that should be able to get your attention immediately, you'll never miss their notifications on your Android phone.

11.4 ● Notifications on receiving MQTT messages

You have been using another notification system from the beginning of this book: MQTT. Maybe you don't think about MQTT as a notification system, but it is: every MQTT client is immediately notified by the MQTT broker of new messages on the topics it's subscribed to. You don't have an MQTT client open all the time, so using MQTT messages as notifications directly will not work. But because the whole architecture of this book's home automation system is centered around MQTT, there's another approach: let a program listen on specific MQTT topics and notify you by email, Gotify push messages, or other means for each received message.

Before you start implementing this idea, let me stop you: Jan-Piet Mens has already created such a system, mqttwarn (https://github.com/jpmens/mqttwarn). It subscribes to any number of MQTT topics and publishes received payloads to one or more notification services after optionally applying some transformations. It comes with over 70 notification handler plugins for a wide range of notification services.

11.4.1 ● Installing mqttwarn

Before installing mqttwarn as a Docker container, create a directory to hold its configuration file:

```
mkdir /home/pi/containers/mqttwarn
```

Then create mqttwarn's configuration file:

```
nano /home/pi/containers/mqttwarn/mqttwarn.ini
```

Enter the basic information for the MQTT connection and logging:

```
; -----------------------------------------
;             Base configuration
; -----------------------------------------

[defaults]

; ----
; MQTT
; ----

hostname     = 'mosquitto'
port         = 1883
username     = home
password     = PASSWORD
clientid     = 'mqttwarn'
lwt          = 'clients/mqttwarn'
skipretained = False
cleansession = False

# Uncomment the following lines for MQTTS and make sure the hostname and port
are correct:
;ca_certs      = '/etc/ssl/certs/rootCA.pem'
;tls_version   = 'tlsv1_2'
;tls_insecure  = False

# MQTTv31 = 3   (default)
# MQTTv311 = 4
protocol     = 4

; -------
; Logging
; -------

; Send log output to STDERR
logfile   = 'stream://sys.stderr'

; one of: CRITICAL, DEBUG, ERROR, INFO, WARN
loglevel  = DEBUG

; name the service providers you will be using.
launch    = log

; -------
; Targets
; -------
```

```
[config:log]
targets = {
    'debug'  : [ 'debug' ],
    'info'   : [ 'info' ],
    'warn'   : [ 'warn' ],
    'crit'   : [ 'crit' ],
    'error'  : [ 'error' ]
  }

)
```

This configures mqttwarn to connect to the mosquitto container on the same Raspberry Pi, unencrypted over Docker's internal network. If you want to connect to an MQTT broker on another machine using TLS, uncomment the lines with `ca_certs`, `tls_version`, and `tls_insecure` and make sure to change the hostname to the fully qualified hostname (for instance `pi-mqtt.home`) and the port to 8883. Don't forget to mount the directory with the root CA file in the Docker Compose file.

Because mqttwarn currently doesn't have a Docker image for ARM yet on the Docker Hub, download its Git repository to build the image yourself:[4]

```
git clone https://github.com/jpmens/mqttwarn
cd mqttwarn
docker build -t mqttwarn .
```

After the Docker image has been built, add the following container definition to your `docker-compose.yml` file:

```
version: '3.7'

services:
  mosquitto:
    # mosquitto config
  mqttwarn:
    image: mqttwarn
    container_name: mqttwarn
    restart: always
    volumes:
      - ./containers/mqttwarn:/etc/mqttwarn
      - /etc/localtime:/etc/localtime:ro
```

4 You can follow the status of an official ARM image on Docker Hub here: https://github.com/jpmens/mqttwarn/issues/424.

Note:
If you want mqttwarn to connect to your MQTT broker with TLS, don't forget to add a line to mount your certificates directory to /etc/ssl/certs.

If by the time you read this book there's an ARM image of mqttwarn on Docker Hub, just use the `jpmens/mqttwarn` image instead in your Docker Compose file.

Now run the container:

```
docker-compose up -d
```

If you look at the logs with `docker logs -f mqttwarn`, you should see a line with "Connected to MQTT broker". This proves that the basic configuration works.

11.4.2 ● Sending emails with mqttwarn

To be able to send emails, you have to add an smtp service configuration at the end of `mqttwarn.ini`:

```
[config:smtp]
server   =  'postfix:25'
sender   =  'MQTTwarn <koen@vervloesem.eu>'
username =  None
password =  None
starttls =  False
targets  = {
    'koen'     : [ 'koen@vervloesem.eu' ],
    'others'   : [ 'other@example.com', 'boss@example.com' ]
    }
```

The server `postfix:25` refers to the postfix container you installed in the previous section with the port number 25. The username and password are empty. Then you can define multiple targets. In this case, the target 'koen' refers to my email address, while the target 'others' refers to a couple of other email addresses.

Now the only thing you should do to make this service active is adding it to the `launch =` line earlier in the configuration. So this line becomes:

```
launch    = log, smtp
```

Now with the email service set up in mqttwarn, you can easily send emails on receiving specific MQTT messages. I'll give an example with a Zigbee contact sensor (the Xiaomi Aqara door and window contact sensor), supported by Zigbee2mqtt (see Chapter 9). When this sensor makes or breaks contact, Zigbee2mqtt sends an MQTT message on the configured topic (for instance `zigbee2mqtt/Front door`) with a JSON payload like the following:

```
{"battery":100,"voltage":3055,"contact":false,"linkquality":52}
```

What if you want to receive an email every time that the status of the sensor changes? You only have to add the following lines to the configuration:

```
[zigbee2mqtt/Front door]
targets = smtp:koen
title = Front door
format = Front door contact changed to {contact}.
```

Then restart mqttwarn:

```
docker restart mqttwarn
```

Now if the contact changes, you receive an email from mqttwarn with the title "Front door" and the message "Front door contact changed to False." when the contact opened, and "Front door contact changed to True." when the contact closed.

11.4.3 ● Transforming and filtering payloads

"Front door contact changed to False." is not a very user-friendly message. Luckily, you can let mqttwarn execute arbitrary Python functions on incoming MQTT messages to transform them before further processing. First, add the following line in the `[defaults]` section of `mqttwarn.ini`:

```
functions = 'funcs.py'
```

Now create this Python file:

```
nano /home/pi/containers/mqttwarn/funcs.py
```

```
"""Custom functions for mqttwarn.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
import json

# Data mapping functions


def translate_xiaomi_aqara_contact(topic, data, srv=None):
    """Translate the Xiaomi Aqara's contact sensor JSON data to a
    human-readable description of whether it's open or closed."""
    payload = json.loads(data["payload"])
    if "contact" in payload:
        if payload["contact"]:
            return dict(status="closed")
        else:
            return dict(status="opened")

    return None
```

This is a data mapping function. Load the MQTT payload as a JSON dictionary and then check whether the "contact" entry is found in the dictionary. If it is, check whether its value is `True` and add an entry `status` with the value `"closed"`. If the value is `False`, add an entry `status` with the value `"open"`.

Now change the configuration in `mqttwarn.ini` to:

```
[zigbee2mqtt/Cabinet door]
targets = smtp:koen
title = Front door
alldata = translate_xiaomi_aqara_contact()
format = Front door {status}.
```

With the `alldata` line you tell mqttwarn to use the specified function to merge the result of the function (a Python `dict`) with the dictionary of the data. This adds a status object that you can then use in the `format` string.

Now restart the mqttwarn container. The result: when your door opens now, you get an email with the message "Front door opened."

Another interesting concept is a filter. Suppose you want to get a notification when your Xiaomi Aqara sensors have a low battery. Zigbee2mqtt sends the battery value as a percentage in the JSON payload, so you only want to get notified when this value is lower than 20. This is easily done by adding the following functions to your `funcs.py`:

```
# Data mapping functions


def zigbee2mqtt_device_name(topic, data, srv=None):
    """Return the last part of the MQTT topic name."""
    return dict(name=topic.split("/")[-1])


# Filter functions


def filter_xiaomi_aqara_battery_low(topic, message):
    """Ignore messages from Xiaomi Aqara when the battery is OK."""
    data = json.loads(message)
    if "battery" in data:
        return int(data["battery"]) > 20

    return True
```

The function `zigbee2mqtt_device_name` is a data mapping function that adds the last part of the MQTT topic as a new item in the processed dictionary, so the device name is more easily accessible in the formatter.

The second function is a filter that returns True when the battery value is higher than 20 and False otherwise.

Now add the following configuration to your mqttwarn.ini:

```
[zigbee2mqtt-battery]
targets = smtp:koen
topic = zigbee2mqtt/+
alldata = zigbee2mqtt_device_name()
filter = filter_xiaomi_aqara_battery_low()
title = {name} battery low
format = {name} has a low battery: {battery}%
```

This shows an alternative way to name your configuration settings. I have used the more

descriptive `zigbee2mqtt-battery` here, which is used as a name instead of an MQTT topic. The topic is specified then in the `topic =` line. In the `alldata =` line you specify the function to execute to enrich the message dictionary. In the `filter =` line you specify which filter is executed on each message. If the filter returns `True` (in this case when the battery value is above 20) the message is discarded. Only if the filter returns `False`, the target is called. Thanks to the title and format strings, you get the message that the battery is low, with the correct name and battery value included.

Just restart the container and you have a low battery notification for your devices.

Note:
If your filter or data mapper function doesn't seem to work, have a look at the output of `docker logs -f mqttwarn`. Chances are that you have some syntax error, and the logs will show you Python's stack trace.

This is just one example of a target and some easy examples for filtering and transforming messages, but I hope it's clear that mqttwarn is a very flexible way to plug other notification systems into the MQTT based home automation system of this book.

11.5 ● Summary and further exploration

Emails are still one of the best ways to send notifications, at least when you don't have to receive the information immediately. In this chapter, you learned how to let the Raspberry Pi OS system services email you, for instance when there are new updates. You also saw a way to let the various Docker containers of your home automation system send you emails.

At the other end of the spectrum, there's the Gotify server that sends push notifications to your Android phone. Every system that can do HTTP requests, including Home Assistant, Node-RED, or your Python code, can notify you instantly with Gotify.

There are still a lot of other notification systems, but actually, you've been using one from the beginning of this book: MQTT. Thanks to mqttwarn you can plug other notification systems into the MQTT based approach of home automation used in this book. If you want to try other notification services with mqttwarn, such as WebSocket or HTTP, read the documentation on mqttwarn's GitHub repository. Moreover, thanks to the possibility to add custom functions, you have the full power of Python at your disposal.

Another interesting option to explore for your home automation system is an SMS gateway, so it can send SMS messages to your phone. This requires a cellular modem with SIM card and some SMS software such as Gammu (https://wammu.eu).

Chapter 12 • Voice control

Voice control is the holy grail of home automation. Science fiction series and movies have accustomed us to spaceships or homes we can talk to. In recent years, voice control at home has become possible thanks to the so-called 'smart speakers' of Google and Amazon.

However, if you think about it, there's nothing smart about these smart speakers: the intelligence is almost completely in the cloud, where your voice recordings are processed and translated into sentences and meaning. This is a complex and very CPU and data-intensive task, and companies like Google and Amazon make us believe that you need the cloud to be able to use voice control. So at first sight it seems that voice control is completely out of reach of a self-hosted home automation system.

Luckily that's not completely true. Granted, you can't have the same performance as those general-purpose smart speakers such as the Google Home and Amazon Echo in a self-hosted system, let alone a Raspberry Pi. But you can have a reasonably working voice control system, even on a Raspberry Pi, if you limit its purpose to some specific domain, for instance, opening and closing your blinds, turning on and off your lights, asking what time it is, and a couple of other specific tasks.

There's a lot of open-source software that implements parts of voice control, but it's a challenge to bring all these parts together and create a working and user-friendly voice control system. A very promising piece of software that is doing exactly this is Rhasspy (https://rhasspy.readthedocs.io), developed by Michael Hansen.[1]

Rhasspy is a voice assistant that works completely self-hosted, is entirely open-source, supports many languages, and works well with Home Assistant and Node-RED. It even runs on the Raspberry Pi. The downside is that you have to train it for your specific use-case.
In this chapter, I show you how to create a voice-controlled home automation system on your Raspberry Pi with Rhasspy.

Warning:
This is the most cutting-edge chapter in this book, and Rhasspy's development is progressing very fast. So some examples in this chapter will not work. Please consult Rhasspy's online documentation when you're stuck.

12.1 • A basic Rhasspy setup

In this section, you set up Rhasspy for voice control on a Raspberry Pi with a microphone and speakers. You can do this on a dedicated Raspberry Pi or your main home automation gateway, as long as it's in a location where you can talk to it and hear its output. Later in this chapter, I show you how you can give Rhasspy a remote 'mouth and ears' by using a second Raspberry Pi.

1 Disclaimer: After I started writing this book, I became involved in Rhasspy's development.

12.1.1 ● Hardware requirements

Rhasspy needs at least a Raspberry Pi 2, but the newer the model the better, especially when you're using more services and a more complex language model. I recommend a Raspberry Pi 4. You also need to connect a microphone and speaker.

For this book, I have tested Rhasspy with the ReSpeaker 2 Mics pHAT from Seeed on a Raspberry Pi 3B and 4B. The pHAT is attached to the Raspberry Pi's GPIO header. You can connect a speaker to the 3.5 mm audio jack or the JST 2.0 connector. I tested this setup with an 8 Ohm 1 W 3" speaker connected to the JST 2 connector of the ReSpeaker. Other Seeed audio devices should work too.

Figure 12.1 A Raspberry Pi with the ReSpeaker 2 Mics pHAT and a speaker (not shown here) is all you need to add voice control to your home automation system.

12.1.2 ● Configure audio hardware

The ReSpeaker 2 Mics pHAT requires you to install a driver:[2]

```
git clone https://github.com/respeaker/seeed-voicecard
cd seeed-voicecard
sudo ./install.sh
```

2 Don't forget to update this regularly with a git pull command in the seeed-voicecard directory and a reinstall of the driver. See Chapter 3.

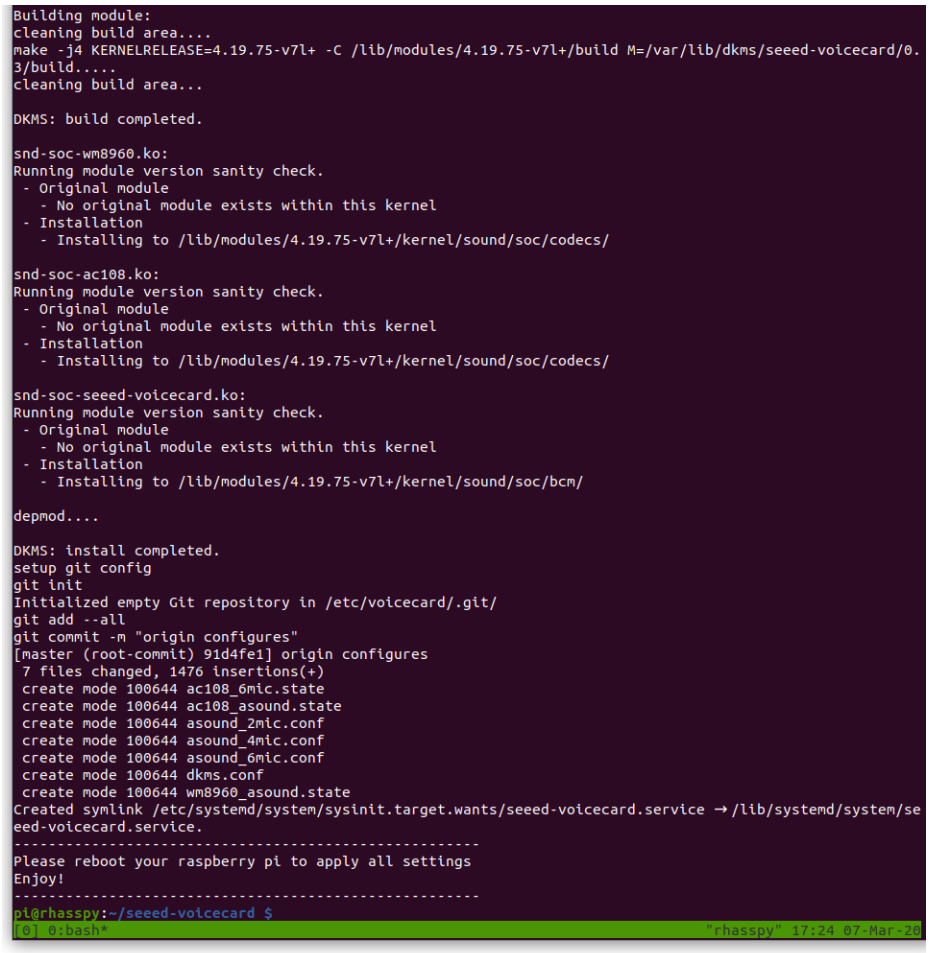

Figure 12.2 The ReSpeaker 2 Mics pHAT requires you to install a driver.

Now to make it easier later to find the right audio output device in Rhasspy's settings, I recommend to disable the Raspberry Pi's on-board audio device:

```
sudo nano /etc/modprobe.d/blacklist-snd_bcm2835.conf
```

List the module to blacklist in this file:

```
blacklist snd_bcm2835
```

And then reboot your Raspberry Pi:

```
sudo reboot
```

After the reboot, check the available audio input devices with:

```
arecord -l
```

This should show the ReSpeaker device.

Now also check the available audio output devices:

```
aplay -l
```

This should only show the ReSpeaker device because the on-board audio device has been disabled.

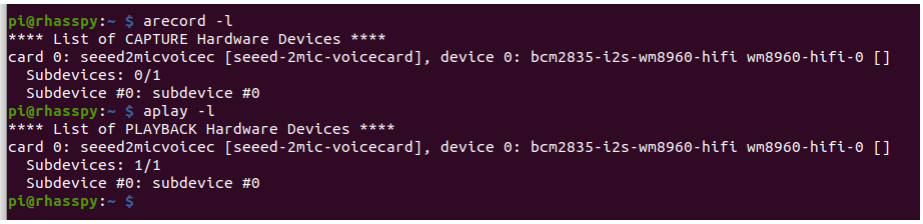

Figure 12.3 After disabling the Raspberry Pi's on-board audio, the ReSpeaker is the only recognized audio device.

You can check whether your audio hardware has been configured correctly by recording a short 5-second audio clip:

```
arecord -f S16_LE -d 5 test.wav
```

Then say something. After five seconds, the recording is saved in the file `test.wav`. Now play this on the Raspberry Pi's speaker with:

```
aplay test.wav
```

If you hear your recording now, your audio setup is ready. Make sure that this works before you install Rhasspy. It makes debugging problems much easier.

12.1.3 ● Installing Rhasspy

Rhasspy can be installed in a Docker container, in a Python virtual environment, or as a Hass.io add-on for Home Assistant. In this chapter, I show you the Docker way.

First, create a directory for the Rhasspy container's configuration and data:

```
mkdir /home/pi/containers/rhasspy
```

Update your `docker-compose.yml` file with:

```
version: '3.7'

services:
  mosquitto:
    # mosquitto config
  rhasspy:
    image: rhasspy/rhasspy
    container_name: rhasspy
    restart: always
    volumes:
      - ./containers/rhasspy/profiles:/profiles
      - ./containers/certificates:/etc/ssl/private:ro
    ports:
      - 12101:12101
    devices:
      - /dev/snd:/dev/snd
     command: --user-profiles /profiles --profile en --certfile /etc/ssl/
private/cert.pem --keyfile /etc/ssl/private/key.pem
```

Make sure to specify the correct file names for your certificate and key files.

Then create the container with:

```
docker-compose up -d
```

This will take a while: the container is more than 400 MB..

When Rhasspy has been started, open its web interface in your web browser by surfing to https://HOSTNAME:12101/ with `HOSTNAME` the hostname of your Raspberry Pi.

12.1.4 ● Rhasspy's settings

The page you're looking at now is the Test page. For now, you can't test anything. The icons on the left bar lead you to other pages, and an alternative way to navigate to other pages is the menu at the top, which shows **Test**. If you click on it, you get a menu with the names of other pages.

Note:
The nice thing about Rhasspy is that it has documentation embedded in its web interface. As long as you haven't configured any Rhasspy services yet, you're greeted by an invitation to have a look at the Getting Started Guide. The settings of the various services also have links to the relevant documentation parts.

Let's go to the **Settings** page first. At the topic, you see a **siteId** (which has the value **default**). This site ID is the name of your Rhasspy device. If you're only using one device with Rhasspy, this name doesn't matter.[3]

Then comes a list of services. As you see, every service there is disabled, except **MQTT** which is **Internal**. That's because Rhasspy by default uses an internal MQTT server that's only available inside its Docker container. However, if you select **External** from the drop-down list, the **MQTT** box becomes green and if you click on it, you can set the host, port, username, and password of your MQTT broker. If you're running Rhasspy from the same Docker Compose file as the mosquitto container, you can enter **mosquitto** as the hostname, **1883** as the port number, and the username and password you configured for mosquitto.

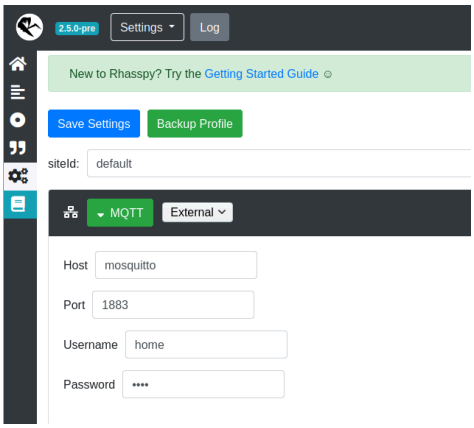

Figure 12.4 Let Rhasspy use the MQTT broker of your home automation gateway.

3 Later you can add satellites, which are other devices that are used for only audio input and/or output and a wake word. They offload the rest of the work to your central Rhasspy device. This makes it possible to put some less powerful Raspberry Pi models around the house that let you talk to the same Rhasspy server everywhere.

To be able to do something with Rhasspy, you have to enable a lot of these services and choose an implementation.

Note:
Rhasspy is more a voice assistant toolkit than monolithic program. For almost any of its services you have the choice between multiple implementations. In this chapter, I make a choice for each of its services, but you can make your own choice after you have experimented with Rhasspy and you have noticed that you have other needs. Rhasspy's architecture is extremely flexible.

12.1.5 ● Configuring audio

Let's first configure the microphone that Rhasspy listens to. Select **PyAudio** next to **Audio Recording**, and click on **Save Settings** below to apply the settings. This will restart Rhasspy.

Afterwards, click on **Audio Recording** to open the audio input settings. If you click on **Refresh**, the input device list becomes populated with your audio input devices. For the Seeed ReSpeaker 2 Mics pHAT, you should see `seeed-2mic-voicecard` in the list. Select it and click again on **Save Settings**.

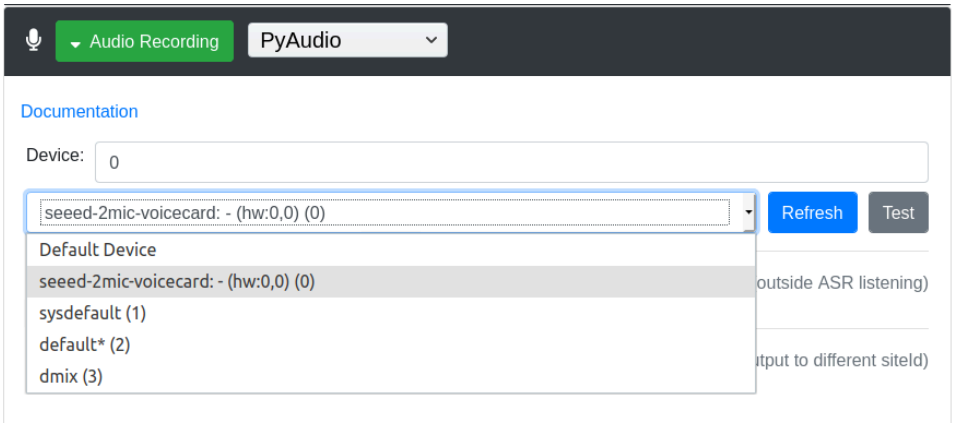

Figure 12.5 Choose your device for audio recording.

Next, you have to choose the device for Rhasspy's audio playback. Go to **Audio Playing** and choose **aplay**. Save the settings and click on the **Refresh** button to refresh the list of available devices and choose your output device. When I selected **Direct hardware device without any conversions**, the device text box is changed to `hw:CARD=seeed2micvoicec,DEV=0`. Now save your settings.

12.1.6 ● Configuring the wake word

With the audio settings configured now, it's time to enable the wake word. This is the word that you use to activate Rhasspy. Choose **Mycroft Precise** in the drop-down list next to **Wake Word** and save the settings.

After this, click on **Wake Word** to return to Rhasspy's wake word settings and click on **Refresh**. You can now choose a wake word from the list of available keywords. The default one is "Hey Mycroft", but some of the other choices are "Sheila", "Athena" and "Marvin". Save the settings after changing the wake word.

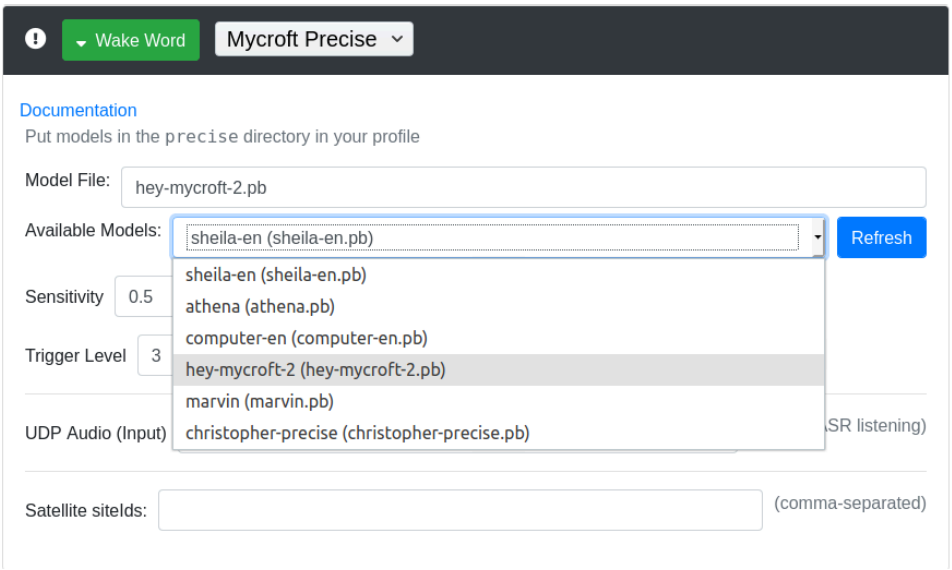

Figure 12.6 Choose your wake word for your voice assistant.

Note:
If the recognition rate of these universal wake words is not enough for you or if you want to use another wake word, you have to train your own wake word, put the resulting model file in Rhasspy's profile directory and refer to it in the wake word settings. Mycroft Precise has some instructions about training a wake word on their wiki: https://github.com/MycroftAI/mycroft-precise/wiki/Training-your-own-wake-word. At the moment this is quite laborious. In the future, Rhasspy will have the training process integrated into its web interface. Another option is to use Porcupine as the wake word engine in Rhasspy, but it has some limits on custom wake words. You could also try Raven, Rhasspy's own wake word system, which is currently in development.

12.1.7 ● Configuring text to speech

When Rhasspy replies to you, it has to convert text to speech. Choose **Espeak** next to

Text to Speech and click **Save Settings.** After this, you can find a list of available voices in the text to speech settings (don't forget to click on **Refresh** first). If you change the default one, click on **Save Settings** again.

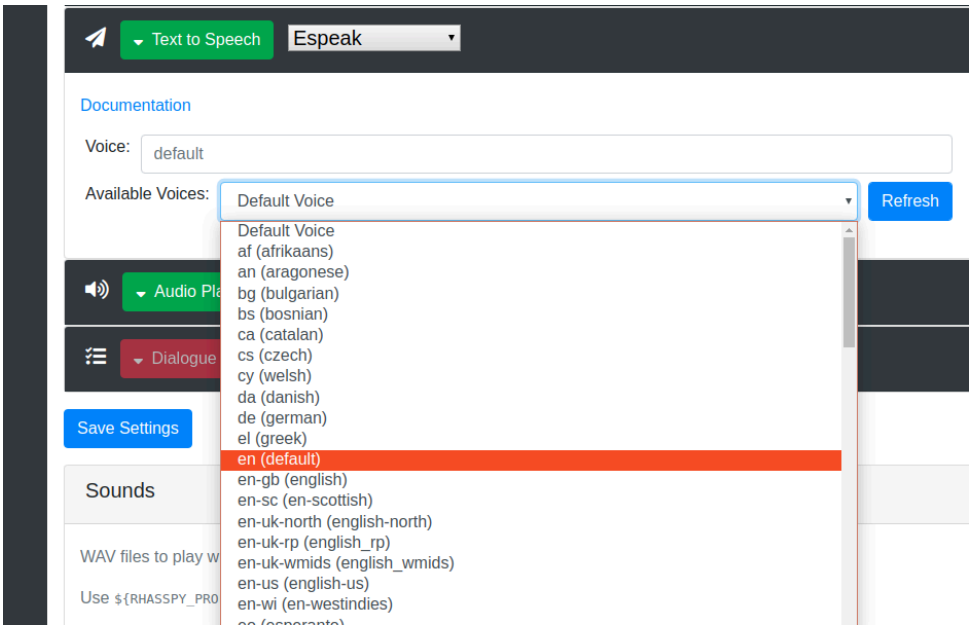

Figure 12.7 You can change Rhasspy's voice to your desired accent.

You can easily test the text to speech. Return to the **Test** page, enter a sentence next to the **Speak** button, and click on the button. You should hear the text you have entered spoken on your Raspberry Pi's speaker.

12.1.8 ● Configuring speech to text

For Rhasspy to be able to understand you, it needs to have a speech to text engine running. Choose **Kaldi** next to **Speech to Text** and then click **Save Settings**.

After Rhasspy has restarted, it warns you that it has to download some files. This is because the speech to text engine uses an acoustic model and base dictionary. Click on **Download**. After all the files have downloaded, your profile is trained and on the bottom, you get a message that training has been completed.

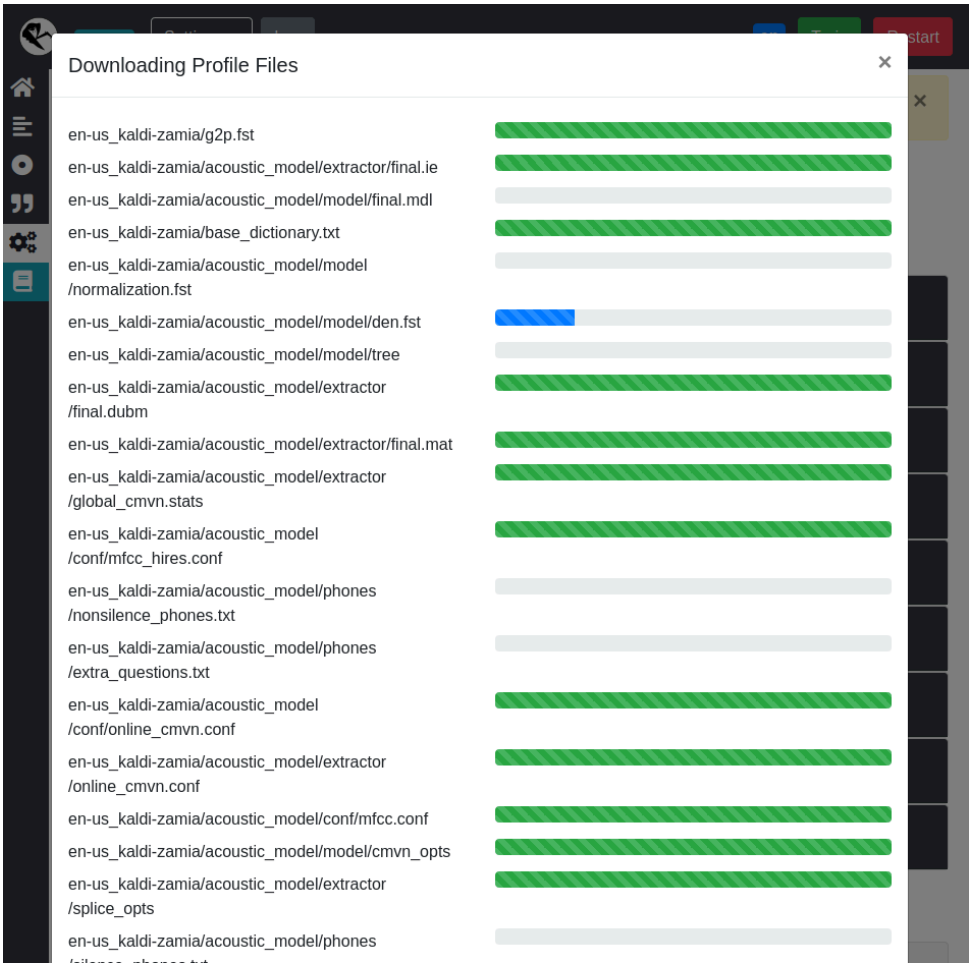

Figure 12.8 Rhasspy needs to download some profile files for Kaldi to work.

12.1.9 ● Configuring intent recognition

Until now you have only been preoccupied with the audio part of Rhasspy: audio in and out, wake word recognition, speech recognition, and speech synthesis. However, you don't only want Rhasspy to know what text is equivalent to the words you spoke, you also want Rhasspy to 'understand' your intent. That's what intent recognition is about.

In Rhasspy's settings, choose **Fsticuffs** next to **Intent Recognition** and click on **Save Settings**. Now Rhasspy can understand your voice commands if they have been defined before. By default, Rhasspy ships with a file with example sentences. You can find it on the Sentences page. For instance, you see an intent GetTime with the example sentences "what time is it" and "tell me the time".

12.1.10 ● Configuring dialogue management

There's one final component you should enable: Dialogue Management. This component manages sessions initiated by a wake word detection or started programmatically by another program. Choose Rhasspy for the dialogue manager.

12.1.11 ● Testing your Rhasspy setup

After all these settings, it's time for a test.[4] So go to the **Test** page, type one of the **sentences** from the Sentences page in the text field that says "Text to recognize" and click on the Recognize button next to it. The page should show **GetTime** in a red box below the button. And when you click on the **Show JSON** button, you see a lot of other information about the intent recognition, such as the recognized tokens (words) and the time needed to recognize the intent.

Intents can also have slots. For instance, if you type "turn on the living room lamp" and click on **Recognize**, you get the intent `ChangeLightState` with two slots shown in blue: `name` with the value "living room lamp" and `state` with the value **on**. If you now click on **Show JSON**, you also get information about the slots.

If this works, try speaking instead of typing your commands. Click on the **Wake Up** button and then speak your command. Rhasspy will show in the "Text to recognize" field which text it has understood from your voice. And if it can link the text to one of its configured intents, you also get the recognized intent.

The final test is waking Rhasspy with your voice instead of the button. Speak your wake word, and when you hear a feedback sound, speak your command. The **Test** page should show your recognized text and intent.

4 Note that I have kept the intent handling component disabled. I'll show later in this chapter how you handle intents using Node-RED or by subscribing to MQTT topics published by Rhasspy. However, you can also define a Home Assistant installation or a HTTP endpoint to handle intents, and even a local program that is called for each intent.

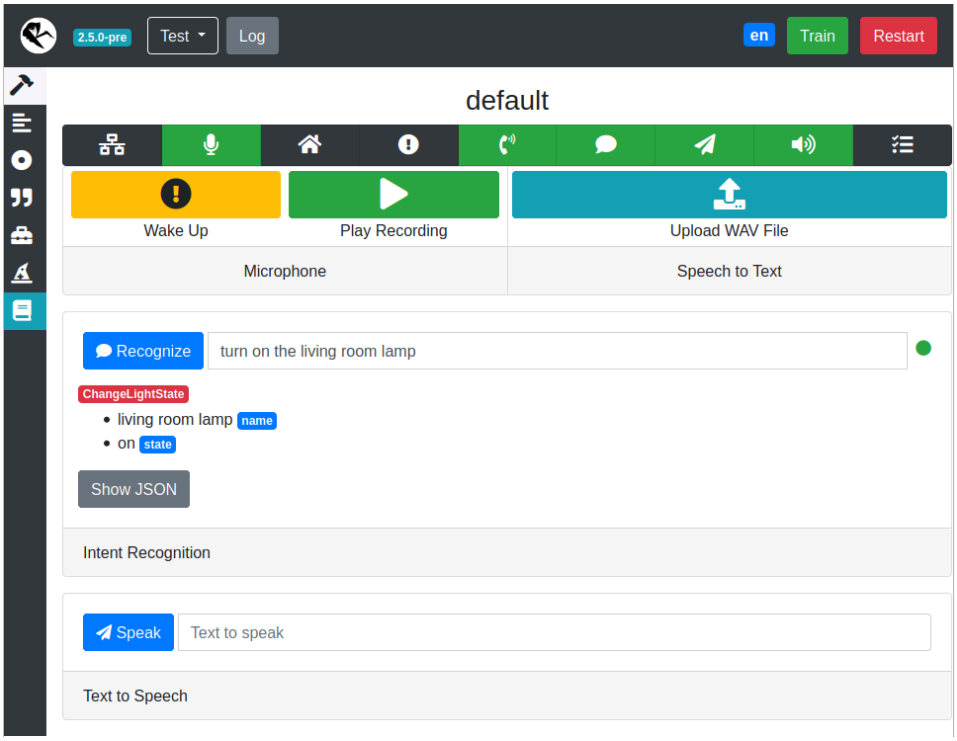

Figure 12.9 If you have configured everything correctly, Rhasspy recognizes that you gave the command to turn on the living room lamp.

If this works: congratulations, you have now an open-source voice assistant running on your Raspberry Pi that works completely offline.

Note:
Rhasspy may have trouble recognizing your commands or your wake word. There's still a lot of tuning you can do and you can use better performing components than the ones I chose in this chapter. A lot of Rhasspy's performance depends on the quality of your audio input and how far you are from the microphone. Rhasspy's documentation has all the information you need and you can ask for help on the forum (https://community.rhasspy.org).

12.2 • A Rhasspy base with satellites

Until now you've been running Rhasspy on just one Raspberry Pi, but what if you want voice control in multiple rooms in your house? That's were a base/satellite (also called master/satellite) setup comes in handy. You run a satellite device in each room and all these satellites communicate with the base.

So instead of running all the services you have configured in the previous section, the base only runs the following Rhasspy services:

- Speech to text
- Text to speech
- Intent recognition
- Dialogue management
- Intent handling (optionally)

The satellites each run the following Rhasspy services:

- Audio recording
- Audio playing
- Wake word

Rhasspy supports two ways of communicating between the base and its satellites: using an MQTT API or a HTTP API. The MQTT API is the most flexible one, and it's a good fit for the approach in this book, so that's what I'll explain in this section.

12.2.1 ● Hardware requirements

The services on the base can be quite CPU intensive if you add complex intents, so it's recommended to use a Raspberry Pi 4 as a base.[5] This Raspberry Pi doesn't need any audio devices for recording and playback, as you'll use the satellites for this purpose. So you can delete the following lines in your `docker-compose.yml` on the base:

```
devices:
  - /dev/snd:/dev/snd
```

Note:
It's perfectly possible to still enable the wake word, audio recording, and audio playback services on your base if you have connected audio devices.

For the satellites, the hardware requirements are much more modest. The only thing the satellites are doing is listening to the audio recording of a microphone, detecting a wake word, streaming audio over the network to the base, receiving audio back from the base, and playing it on the speaker. A Raspberry Pi Zero W or any of the older Raspberry Pi models should be up to the task, depending on the wake word component you run. You need supported audio hardware on the satellites, such as the ReSpeaker 2 Mics pHAT and a speaker.

5 If your intents are very complex or if you want to use alternative Rhasspy components that are better but slower, even a Raspberry Pi 4 can't handle the task. You should use an Intel NUC (Next Unit of Computing) or server with Linux for the base then.

Figure 12.10 A Raspberry Pi Zero W with the ReSpeaker 2 Mics pHAT and a speaker is the perfect satellite device for Rhasspy.

12.2.2 ● Setting up the satellites

On each satellite device, make sure that you configure and test the audio hardware and then install Rhasspy as explained in the previous section.

Go to the **Settings** page, and enter a unique name for the **siteId**. This should probably be a description of the location of your satellite, such as **kitchen, livingroom,** or **office**.

Then set **MQTT** to **External** and change the host, port, username, and password to the relevant settings of your MQTT broker.

Configure **Audio Recording, Wake Word**, and **Audio Playing** as in the previous section. Leave **Intent Handling** and **Dialogue Management** disabled.

With these settings, your satellite can record and play audio and detect its wake word, but it doesn't do anything with it. So the next step is to set **Speech to Text, Intent Recognition** and **Text to Speech** to **Hermes MQTT**. This makes Rhasspy on the satellite use MQTT messages with the Hermes protocol for these services.

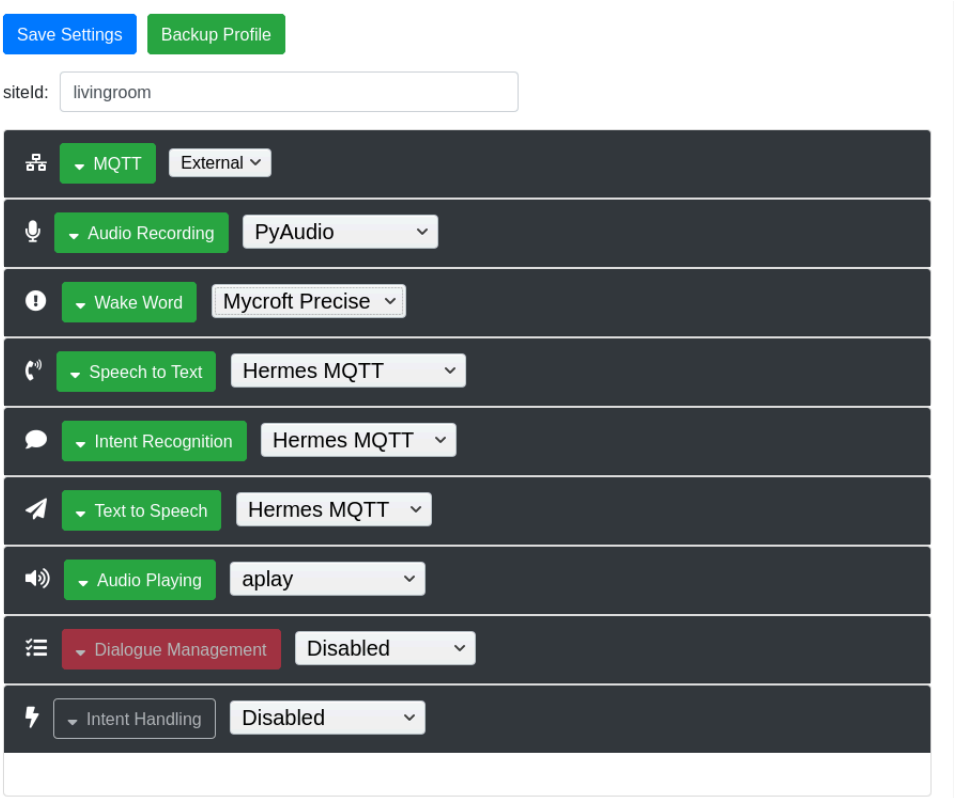

Figure 12.11 The satellite offloads speech to text, intent recognition, and text to speech to the base using the Hermes MQTT protocol.

After all these changes, click on Save Settings. Now repeat the same process on all your other satellites, but make sure that each satellite has a different site ID.

12.2.3 ● Setting up the base

After you have set up your satellites, it's time to configure the base system. Go to the **Settings** page and enter a unique name for the **siteId**. If you don't use a microphone and speaker on this base, you don't have to describe the location in its name. Just use **base** or **master** then.

Then set **MQTT** to **External** and change the host, port, username, and password to the relevant settings of your MQTT broker. If you're running your MQTT broker on the same Raspberry Pi in the `mosquitto` container, you can just use `mosquitto` as the hostname.

Now leave **Audio Recording, Wake Word** and **Audio Playing** disabled, as these are handled by your satellites. You can leave **Intent Handling** also disabled for now; I'll show another way to handle intents later in this chapter.

The next step is to enable your **Speech to Text, Intent Recognition,** and **Text to Speech** services. Configure them like in the previous section. Also set **Dialogue Management** to **Rhasspy**. Now for each of these services, add the siteIds of all your satellites in the **Satellite siteIds** text field, separated by a comma, for instance, **livingroom,kitchen,office.**

After all these changes, click on **Save Settings**. After Rhasspy has restarted on your base, your setup is ready for a test.

The dialogue manager makes the base listen and respond to requests for each satellite. It gets notified on wake word detections on each satellite, it engages the speech recognition and intent recognition, and it can generate audio with the speech synthesis and relay it back to the right satellite to play it on its speakers, all using MQTT messages.

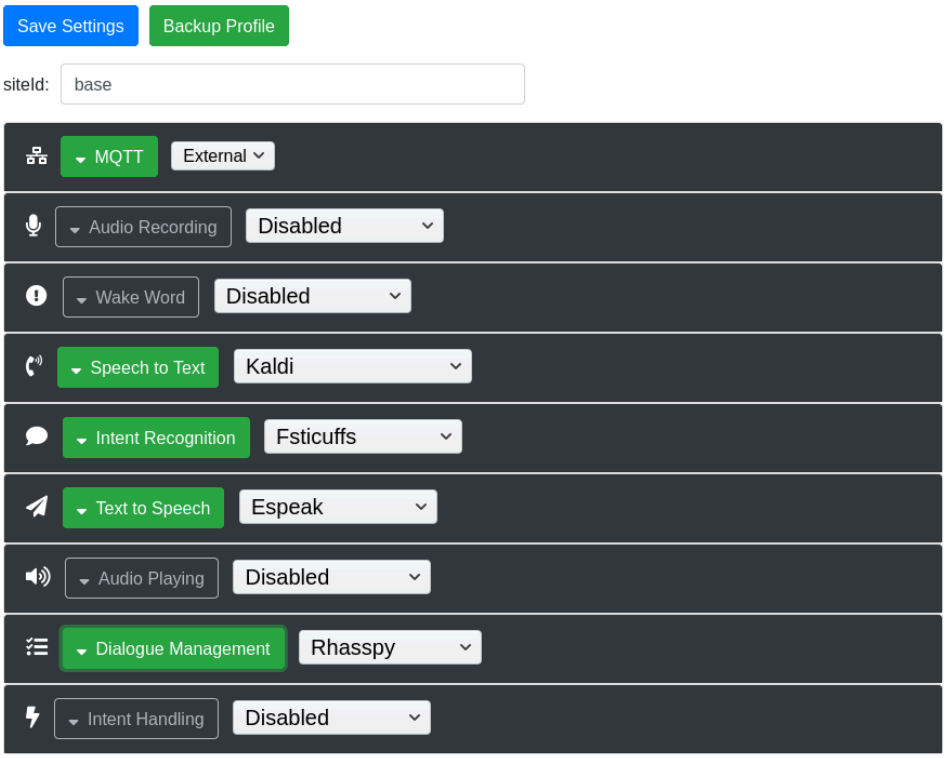

Figure 12.12 The base doesn't need audio and wake word services.

12.2.4 ● Testing your base and satellites

Wake up one of your satellites with its wake word. Then speak your command, such as "turn on the living room lamp". The audio stream will be sent from the satellite to your base, where it's transcribed thanks to the speech to text engine. If all goes well, the intent is recognized, which you'll see on the **Test** page.

You can also check this with the Mosquitto client on the command line:

```
mosquitto_sub -t 'hermes/nlu/#' -v
```

Now the text recognized in your voice is sent to the `hermes/nlu/query` topic, with a JSON dictionary as its payload, for instance:

```
{
   "input": "turn the bedroom light on",
   "intentFilter": null,
   "id": "",
   "siteId": "livingroom",
   "sessionId": "livingroom-default-171e8db6-e5d7-4f8c-ac4e-c67dac27377c"
 }
```

The intent recognition service uses this as its input, and if it recognizes intent in this text, it sends a JSON payload to the `hermes/nlu/intentParsed` topic, such as:

```
{
  "input": "turn the bedroom light on",
  "intent": {
    "intentName": "ChangeLightState",
    "confidenceScore": 1
  },
  "slots": [
    {
      "entity": "name",
      "slotName": "name",
      "confidence": 1,
      "raw_value": "bedroom light",
      "value": "bedroom light",
      "range": {
        "start": 9,
        "end": 22,
        "raw_start": 9,
        "raw_end": 22
      }
    },
    {
     "entity": "name",
      "slotName": "name",
```

```
      "confidence": 1,
      "raw_value": "on",
      "value": "on",
      "range": {
        "start": 23,
        "end": 25,
        "raw_start": 23,
        "raw_end": 25
      }
    }
  ],
  "id": "",
  "siteId": "livingroom",
  "sessionId": "livingroom-default-171e8db6-e5d7-4f8c-ac4e-c67dac27377c"
}
```


You can see a lot of information here: the text input, the recognized intent (`ChangeLightState`), the slots and their values, and even the `siteId` of the satellite where you spoke your command.

Note:
There's a lot more data sent around on MQTT. Have a look at the complete reference of the MQTT API (https://rhasspy-hermes.readthedocs.io/en/latest/api.html).

12.2.5 ● Enable UDP audio streaming

By default, satellites continuously stream recorded audio over MQTT. This can strain your network if you have a lot of satellites, and it's also not nice from a privacy point of view. You can check this by listening to the MQTT topics `hermes/audioServer/#`. You'll see a continuous stream of garbled data. These are the RAW audio recordings from the microphones of your satellites.

That's why Rhasspy has another option: let the audio recorder on a satellite stream audio locally over a UDP port to the wake word engine on the same machine, and only start streaming audio over MQTT to the rest of the network once a wake word is detected and until your command is finished.

To use this option, open the **Settings** page on each satellite, open **Audio Recording**, and enter a port number in the **UDP Audio (Output)** port field, for example **12202**. You can keep the host field empty. Now enter the same port number in the **UDP Audio (Input)** field in the **Wake Word** section and save your settings.

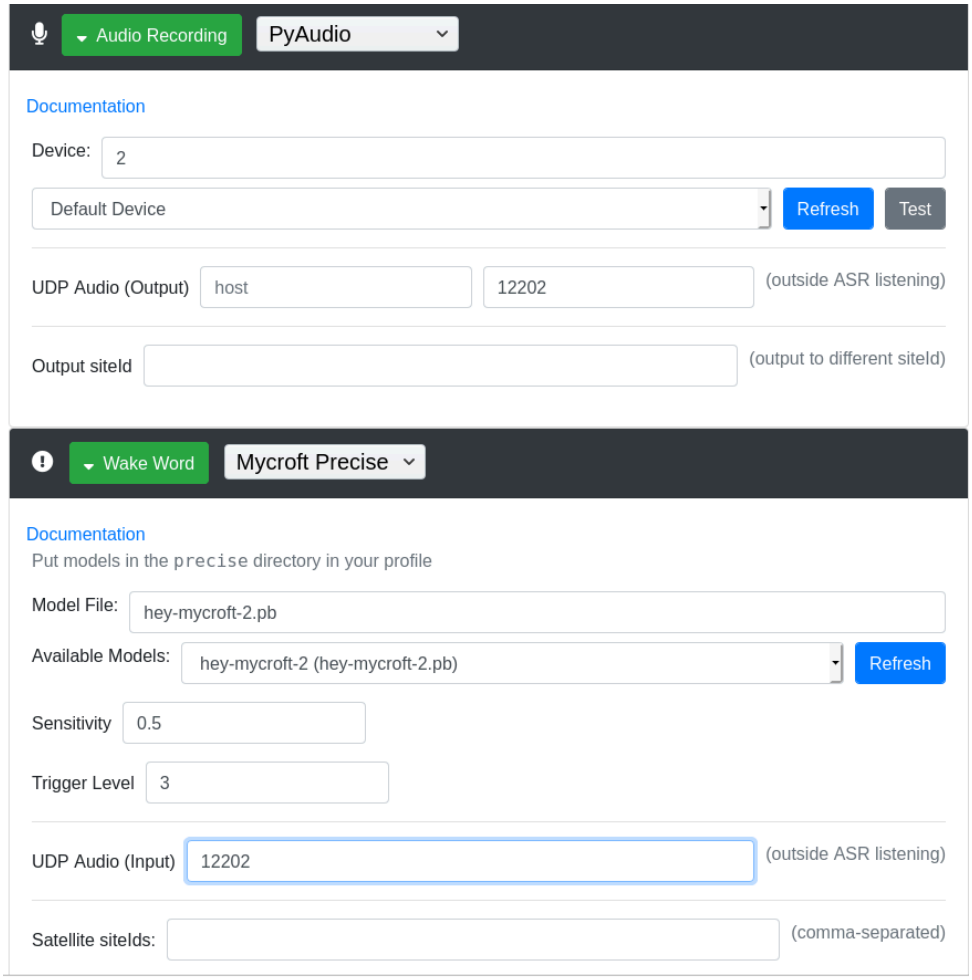

Figure 12.13 With UDP audio streaming the audio recorder doesn't continuously stream audio data over MQTT.

After this change, you should still be able to test Rhasspy successfully, but audio is only streamed from your satellite to your base when it's listening for a command after it has detected its wake word.

12.3 • Train your sentences

By default, Rhasspy has defined a couple of intents, each with their example sentences. You can find these on the **Sentences** page, formatted as an ini file. The default file looks like this:

```
[GetTime]
what time is it
tell me the time

[GetTemperature]
whats the temperature
how (hot | cold) is it

[GetGarageState]
is the garage door (open | closed)

[ChangeLightState]
light_name = ((living room lamp | garage light) {name}) | <ChangeLightColor.
light_name>
light_state = (on | off) {state}

turn <light_state> [the] <light_name>
turn [the] <light_name> <light_state>

[ChangeLightColor]
light_name = (bedroom light) {name}
color = (red | green | blue) {color}

set [the] <light_name> [to] <color>
make [the] <light_name> <color>
```

Each intent has a section in the ini file. For instance, the first section in the file is designated by the `[GetTime]` section header, which lists all example sentences that will be assigned to the `GetTime` intent. So if Rhasspy hears you say "what time is it" or "tell me the time", it will recognize the `GetTime` intent.

Note:
Don't use non-word characters, such as punctuation marks. Even question marks are ignored. Capitalization is ignored too.

Every time you change something in the text field with your intents, you should click the **Save Sentences** button. This also re-trains your language profile, so Rhasspy recognizes your new sentences.

Note:
If you add words to your sentences that Rhasspy doesn't know how to pronounce, this will be shown on the **Words** page. You have to add the phonetic pronunciation for every unknown word there.

You can also add new sentences files by clicking on **New Sentences File** and choosing a file name. You can select the sentences file you want to edit. This way you don't have to add all your intents in one file.

12.3.1 ● Rhasspy's template language

The example of the `GetTime` intent is rather basic: you just add some sentences and when you say one of these sentences verbatim, the corresponding intent is recognized. However, Rhasspy can do a lot more: it has a template language so not every sentence for an intent has to be spelled out verbatim.

You can have alternatives, such as in the sentence `how (hot | cold) is it`. This sentence template matches both sentences "how hot is it" and "how cold is it". You can also add optional words, such as in `turn [the] light on`, which matches both sentences "turn the light on" and "turn light on".

The more complex your intents become, the more you want to reuse parts of your sentence templates. For instance, let's say you have an intent to set the light to a color:

```
[SetLightColor]
set the light to (red | green | blue)
```

Now you want to add an intent to ask what the current color of the light is:

```
[GetLightColor]
is the light (red | green | blue)
```

This works, but now your colors are duplicated in two intents. And if you want to support a new color, you have to change the colors in both places.

That's where rules come into play. You define a rule in one of the intents, and in the sentence, you refer to the name of this rule, which is substituted by the colors. Even more, you can also refer to this rule in other intents. The result is that you don't have to duplicate all these colors anymore:

```
[SetLightColor]
colors = (red | green | blue)
set the light to <colors>

[GetLightColor]
is the light <SetLightColor.colors>
```

Now if you add a color in the rule `colors = (red | green | blue)` of the `SetLightColor` intent and retrain Rhasspy, the new color is automatically recognized in the `GetLightColor` intent too.

There's a lot more possible with Rhasspy's template language, but you can go a long way with these basics. Consult the online documentation for the full syntax.

12.3.2 ● Slots

Colors can easily be added as alternatives in a rule, but if a list of alternatives becomes too long, it's better to use slots for it. Go to the page **Slots** and then click the **New Slot** button and choose a slot name, for instance, `rooms`.

Now you can add a room on each line in the text field, for example:

```
bathroom
(living room):living_room
office
kitchen
toilet
bedroom
```

Note the line `(living room):living_room`. This will substitute "living room" by "living_room" in the rooms slot. You can use this to replace a user-friendly name by a name more suited for one of the programs that will process your intents.

So how do you use these slots? If you have saved the slot as rooms, you can refer to it in your sentence templates as `$rooms`, for instance:

```
[TurnOnLight]
turn the ($rooms){room} light on
```

After retraining Rhasspy, the command "turn the living room light on" will match the `TurnOnLight` intent, because "living room" is in the `rooms` slot.

Moreover, if you look at the JSON payload of the intent, you see that the `room` slot has the value `living_room`. The slot is called `room` because the `{room}` tag has been attached to the `($rooms)` slot in the example sentence. And the value is `living_room` because in the slot list `living room` is replaced by `living_room`. This is just one example of the very powerful syntax of Rhasspy's templates.

12.4 ● Intent handling

Until now Rhasspy just showed what intent it has recognized, but it doesn't do anything with it. That's the last step in your voice control system: intent handling.

Maybe you remember that you have kept the **Intent Handling** component disabled in Rhasspy's settings. If you click on the dropdown list, you'll see that Rhasspy can use Home Assistant, a remote HTTP server or a local command to do intent handling. However, you don't need to do this, you don't even have to enable intent handling, because recognized intents are already sent as MQTT messages to your MQTT broker using the Hermes protocol, and available as events on a WebSocket connection. In this section, I show you both ways to handle intents.

12.4.1 ● Intent handling with MQTT

You can easily create your own intent handler, for instance with a Python program using the Paho MQTT library. See Chapter 4 for some background. Here I show you a simple example of a Python intent handler for Rhasspy, using the room slots and `TurnOnLight` intent I defined in the previous section.

```
"""Rhasspy intent handler for the TurnOnLight intent.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
import json

import paho.mqtt.client as mqtt

MQTT_HOST = "pi-red"
MQTT_PORT = 1883
MQTT_CLIENT_ID = "Lights"
MQTT_USERNAME = "home"
MQTT_PASSWORD = "PASSWORD"

INTENT_LIGHT = "hermes/intent/TurnOnLight"
INTENT_NOT_RECOGNIZED = "hermes/nlu/intentNotRecognized"
```

```
TTS_SAY = "hermes/tts/say"


def on_connect(client, userdata, flags, rc):
    """Subscribe to the right MQTT topics after connecting."""
    print("Connected with result code " + str(rc))
    client.subscribe(INTENT_LIGHT)
    client.subscribe(INTENT_NOT_RECOGNIZED)


def on_message(client, userdata, msg):
    """Called each time an intent is recognized (or not)."""
    nlu_payload = json.loads(msg.payload)
    if msg.topic == INTENT_NOT_RECOGNIZED:
        sentence = "Unrecognized command."
        print("Recognition failure")
    else:
        print("Got intent:", nlu_payload["intent"]["intentName"])
        room_slot = nlu_payload["slots"][0]
        room_name = room_slot["raw_value"]
        sentence = "Turning on {} light.".format(room_name)

    site_id = nlu_payload["siteId"]
     client.publish(TTS_SAY, json.dumps({"text": sentence, "siteId": site_
id}))


if __name__ == "__main__":
    # Initialize MQTT connection
    mqtt_client = mqtt.Client(MQTT_CLIENT_ID)
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message
    mqtt_client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)
    mqtt_client.connect(MQTT_HOST, MQTT_PORT)

    # Start event loop
    mqtt_client.loop_forever()
```

The basic structure of this program is the same as in Chapter 4, with `on_connect` and `on_message` callbacks.

The `on_connect` function is called when connecting to the MQTT broker. The program subscribes then to two MQTT topics: one for the intent `TurnOnLight` and one that Rhasspy publishes to when there's no intent recognized.

The `on_message` function is where the magic happens. This function is called when Rhasspy publishes to one of the topics this program has subscribed to. If the intent is recognized, its JSON payload is decoded, and some information is extracted, such as the intent name and the raw value of the room slot.[6]

At the end of the function, a reply is published to the `hermes/tts/say topic`. This is a JSON object with two elements: the text to say and the siteId where Rhasspy has to say the text. This siteId can be extracted from the original intent.[7]

If you run this Python program in a virtual environment with `paho-mqtt` installed and then say "turn the living room light on", Rhasspy recognizes the `TurnOnLight` intent and publishes it on MQTT. The program receives the intent in an MQTT message, extracts the slot name, and replies that it turns that light on.

Note:
Replying to your command is the only thing this Python program does. I'll leave it as an exercise for you to expand upon and let the program turn your living room light on. With the knowledge of the previous chapters, it should be an easy task to add this.

12.4.2 ● Intent handling with AppDaemon and MQTT

The Python script in the previous subsection is a simple example of intent handling, but if you want to have multiple Python scripts handling various intents, they all have to define the MQTT host and port, username and password, initialize the MQTT connection and start the event loop. As you have seen in Chapter 10, a better solution, in that case, is AppDaemon.

Let's see how an AppDaemon app for Rhasspy would look like. First, create a directory for the app:

```
mkdir /home/pi/containers/appdaemon/apps/whatsthetime
```

Then the app's code looks like this:

6 The raw value of a slot (`raw_value`) is the value as it's spoken in the recognized sentence, for instance "living room". The value of a slot `(value)` is the same as the raw value except when it's substituted by a rule. In this example the value would be "living_room".

7 Make it a habit to always add punctuation to the sentences you send to the text to speech engine, because some TTS systems have a problem without it. That's why I added the period at the end of the sentences. I haven't tested this, but Michael Hansen warned me that without this period the voice that MaryTTS (http://mary.dfki.de) generates sounds like it's having a stroke.

```
"""Tell the time when someone asks for it using Rhasspy's Hermes API.

Copyright (C) 2020 Koen Vervloesem

License: MIT
"""
import json

import mqttapi as mqtt

MQTT_MSG = "MQTT_MESSAGE"
INTENT_GETTIME = "hermes/intent/GetTime"
TTS_SAY = "hermes/tts/say"
TIME_FORMAT = "%H %M"


class WhatsTheTime(mqtt.Mqtt):
    """Rhasspy app that tells the time."""

    def initialize(self):
        """Initialize the app by subscribing to the right intent."""
        self.set_namespace("mqtt")
       self.listen_event(self.on_time_request, event=MQTT_MSG, topic=INTENT_
GETTIME)
        self.log("What's The Time app initialized")

    def on_time_request(self, event_name, data, kwargs):
        """Tell the time."""
        nlu_payload = json.loads(data["payload"])
        site_id = nlu_payload["siteId"]
        now = self.datetime().strftime(TIME_FORMAT)
        sentence = f"It's {now}."
         self.mqtt_publish(TTS_SAY, json.dumps({"text": sentence, "siteId":
site_id}))
```

You see that there's much less boilerplate code. On initialization, the app subscribes to the hermes/intent/GetTime MQTT topic. When Rhasspy recognizes the GetTime intent in your speech, a message is sent to this MQTT topic and the on_time_request method is called. The MQTT message's payload is decoded as a JSON dictionary, and the siteId is extracted from the MQTT message.

Then the current time is put in a sentence, and this sentence is put in a JSON dictionary together with the right siteId. The result is published to the `hermes/tts/say topic`, and Rhasspy tells you the time.

Save this file as `whatsthetime.py` in the `apps/whatsthetime` directory of AppDaemon, and also create a file called `whatsthetime.yaml` in the same directory. Give this file the following content:

```
whatsthetime:
  module: whatsthetime
  class: WhatsTheTime
```

After you have saved both files, AppDaemon automatically picks up the changes and loads your app. If you now ask Rhasspy "What's the time", it will answer you with the current time.

12.4.3 • Intent handling with WebSocket in Node-RED

If you prefer a graphical way to program your intent handling, Node-RED is the perfect solution. In Chapter 10 I already showed you how you could subscribe to MQTT topics. Sending MQTT messages is as straightforward. So you could perfectly create a Node-RED flow to handle Rhasspy's intents using MQTT.

However, just to illustrate Rhasspy's versatility and the power and simplicity of its WebSocket API, I show you here how to create a Node-RED flow talking to Rhasspy using the WebSocket protocol.

Create a new flow in Node-RED by clicking on the + icon at the top right. Double-click on the name of the flow to edit it and call it **Rhasspy**.

Pick a **websocket in** node from the **network** category and drag it to the canvas. Double-click on it, leave the type on **Listen on**, and click on the pencil icon next to **Add new websocket-listener...** for the **Path** field. Enter **ws://rhasspy:12101/api/events/intent** in the **Path** field: this is the WebSocket URL for Rhasspy's API.

Note:
The hostname rhasspy refers to the name of the container that is running Rhasspy. If Node-RED and Rhasspy are running on different machines, you need to enter another hostname here.

Change **Send/Receive** from **payload** to **entire message** and click **Add** to add this WebSocket listener. Then you're back in the settings of your **websocket in** node. Give it a name (for instance **Rhasspy**) and click **Done**.

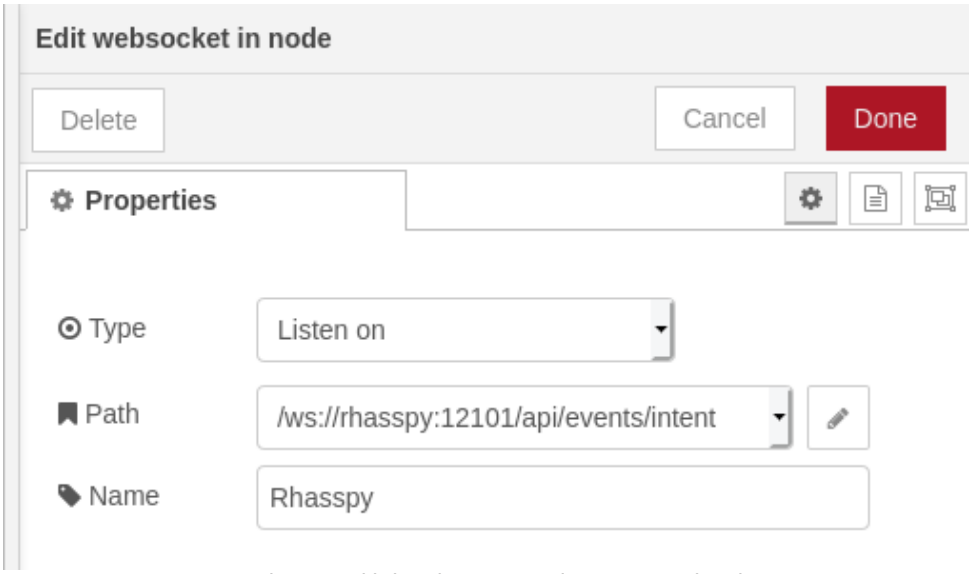

Figure 12.14 Rhasspy publishes the recognized intents as WebSocket events.

Drag a **debug** node to the canvas and link its input to the output of the websocket in node. Deploy your flow.

Now click on the hamburger menu at the top right and then choose **View** and then **Debug messages**. If you now talk to Rhasspy and it recognizes an intent, Node-RED shows the intent as a JSON object in the debug panel at the right.

Let's now add some action to the flow. Drag a **switch** node (from the **function** category) to the canvas and link it to the output of the **websocket** in node. Double-click on the switch node and change the **Property** field to **msg.intent.name**. Leave the rule with **==** empty: this way you check for an unrecognized intent (because the intent then has an empty name). Click **Done** to save the node.

Then add a **change** node (also from the **function** category) and link it to the switch node's output. Double-click on the change node and set the **msg.payload** to "What did you say?". Give the node a descriptive name and click on **Done**.

Then put a **http request** (from **network**) node on the canvas and link it to the change node. Double-click on the node, change the **Method** to **POST**, the **URL** to **http://rhasspy:12101/api/text-to-speech**, and give the node a name. If you deploy your flow now and tell Rhasspy something it doesn't recognize, Node-RED asks you "What did you say?".

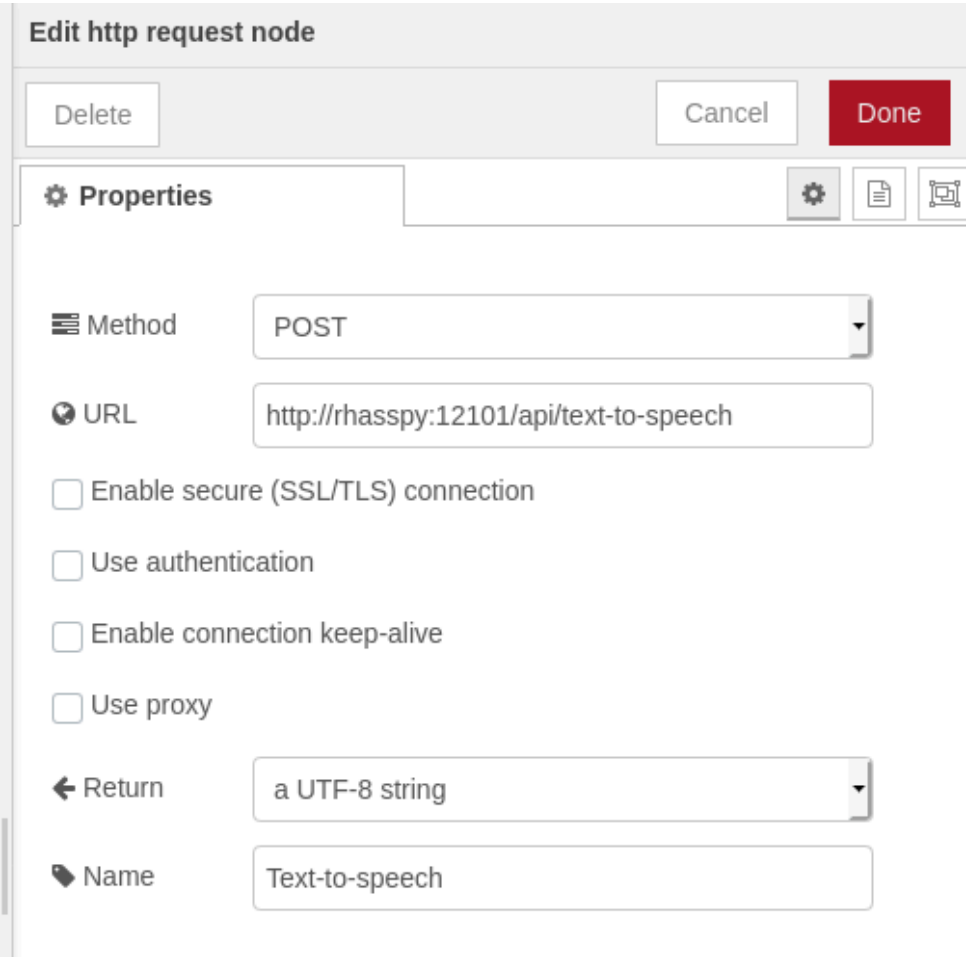

Figure 12.15 You can ask Rhasspy to speak a sentence by doing a HTTP POST request.

Now you can easily add other rules to the switch node to reply to other recognized intents. For instance, to reply to the `GetTime` intent, double-click on the **switch** node, click on the **+add** button at the bottom to add a rule and then enter **GetTime** in the field next to the double equals sign.

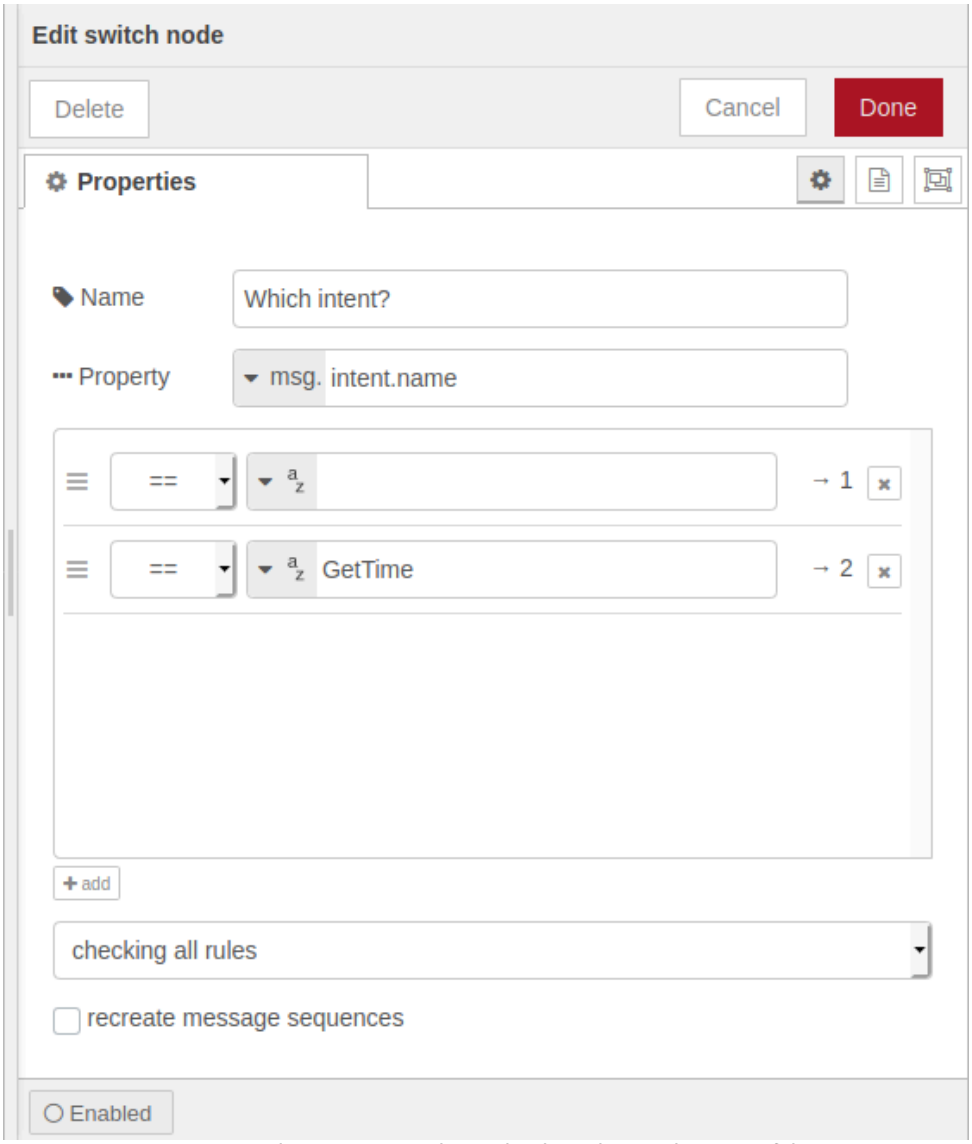

Figure 12.16 Let Node-RED execute other nodes depending on the name of the intent.

After you have saved the switch node, a second output appears. Now drag a **function** node (in the **function** category) to the canvas and link its input to this second output. Also, link the output of this function node to the http request node. Double-click on the function node to open its properties.

A function node lets you write JavaScript code to do more complex manipulation than is possible with the standard nodes in Node-RED. And some tasks are possible with Node-RED nodes but easier in plain JavaScript.

In this case, I use JavaScript to get the current time and put it in a text to send to Rhasspy's text to speech engine. The code looks like this:

```
var timeString = new Date().toLocaleTimeString([],
{
    hour: "2-digit",
    minute: "2-digit",
    hour12: false
})

return {
    payload: "It's " + timeString.replace(":", " ")
}
```

First I create a time string, such as "09:48", for the current time. Then I return this string, preceded by "It's " and with the colon replaced by a space for easier pronunciation.

Note:
There's still room for improvement. Rhasspy speaks each number separately, for instance "zero nine four eight". This is an exercise for the reader.

After saving this function node, deploy your flow. If you ask Rhasspy "What time is it?", your Node-RED flow reacts by sending the current time to Rhasspy, which tells you the time.[8]

The resulting flow looks like this and can be easily extended for other intents:

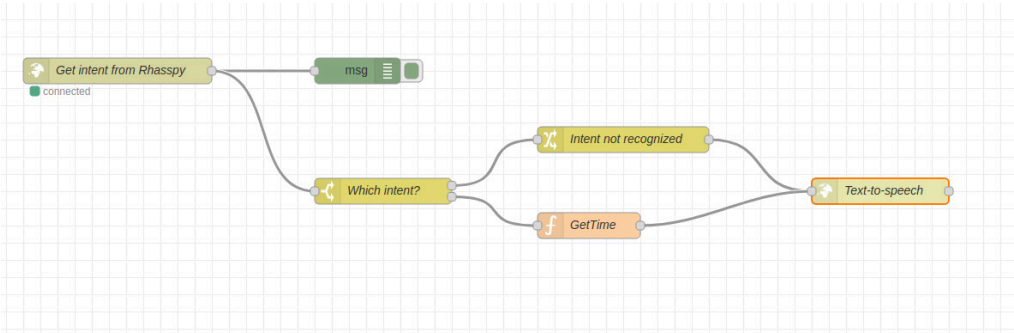

Figure 12.17 Node-RED lets you react to Rhasspy's intent recognition.

8 Make sure to stop the AppDaemon app from the previous subsection that reacts to the same intent if you test this Node-RED flow.

12.5 ● Summary and further exploration

As is clear from this chapter, voice control is quite a complex task consisting of various closely interrelated components. Luckily Rhasspy integrates all these components in one web interface and lets them talk over MQTT using the Hermes protocol. This makes it easy to create one "base" system that does the heavy lifting and connect multiple "satellites" with audio hardware that you talk to.

In contrast to the well-known cloud-based voice assistants, on the Raspberry Pi, it's not yet possible to recognize unrestricted voice commands. The processing power needed to understand you as if you are talking to another human is just too much.

However, you probably don't want to talk to your Raspberry Pi as if it's another human: you want to give it specific commands and ask it for specific information. In this chapter, I have shown you how you can train your sentences so Rhasspy can recognize intent. And because these sentences and intents are quite limited in number, your Raspberry Pi can understand them very well with limited processing requirements.

If there's one chapter where you can keep exploring more and more, it's this one. Voice technology is a fascinating domain of research, and a lot of open-source projects let you experiment with them. You can swap one of the components I used in this chapter for another one to see whether it performs better for you. For instance, I recommend you to explore MaryTTS (http://mary.dfki.de) as the text to speech engine, because it sounds much better than the rather robotic voice of eSpeak. Michael Hansen has even created a Docker image so you can easily run MaryTTS, also on your Raspberry Pi (https://github.com/synesthesiam/docker-marytts). Try swapping Fsticuffs for Rasa NLU (https://rasa.com/docs/rasa/nlu/about/) as the intent recognizer. It's also very enjoyable to develop complex intent handling scripts to make every aspect of your home automation system controllable by your voice.

Writing apps that can react to Rhasspy's intents is not yet as developer-friendly as it should be, but that's because the focus of Rhasspy's development has been on the lower-level parts of the voice technology stack: getting all these components to work together. There will probably appear some helper libraries and tools to make this easier, with much less boilerplate code. When I was finishing the book, there were already some tentative discussions in Rhasspy's forum, and I created my own helper library (https://rhasspy-hermes-app.readthedocs.io). Have a look there if you want to write the next killer app for voice control.

If you're not so much interested in a complete voice assistant for your home automation setup, but if you're looking for a way to quickly add speech recognition to an existing program or to do an offline batch process of speech/intent data, have a look at another project of Rhasspy's creator, voice2json (https://voice2json.org). You can use it to launch programs on your computer, set timers, and so on.

When I asked Michael Hansen to review this chapter, he added:

'Rhasspy's (and voice2json's) profiles come from a curated collection of free models (https://github.com/synesthesiam/voice2json-profiles/) that were made possible by donating their data. I think the future of offline voice assistant technology is going to depend on finding ways to collect large amounts of (donated) data without invading people's privacy like Google/Amazon/etc. do.'

I agree with him that this will be the next big challenge for self-hosted voice assistants. We have all the technology, but not the data.

Rhasspy is not the only project that is creating an open-source voice assistant, but it's currently the most promising one for a self-hosted setup. Another interesting project is Mycroft AI (https://mycroft.ai).[9] However, at the moment it still sends your voice commands to a cloud service to recognize them.

9 The wake word system Mycroft Precise that you configured earlier in this chapter is a part of the Mycroft AI project.

Chapter 13 ● Remote access

Automating and controlling devices at home is nice, but you probably also want to have access from remote locations. For instance:

- You're at work and there's a storm outside. You want to look at the camera feed from your garden to check whether there's some damage from fallen trees.
- You're at a club with friends and you're staying longer than expected. You want to be able to close your motorized blinds at home remotely.

Many commercial home automation systems implement this remote access by having communication from all users relayed by a central server. You log into a web interface from the company, and on this web interface, you have access to your home automation gateway at home.

With a self-hosted home automation system like the one that I implement in this book, you don't need to rely on a third party.[1] In this chapter, I'll show you how you can remotely access your home automation system securely, with the least reliance on other parties as possible.

Warning:
There's a reason why this chapter is at the end of the book and the chapter about security is at the beginning. From the moment you give remote access to your home automation system, it's very important to have a secure setup. If not, malicious people can abuse your system and make your life very miserable. So before you start implementing the remote access solutions in this chapter, revisit Chapter 3.

13.1 ● Three ways for remote access

There are a couple of ways to implement remote access to your home automation gateway (or any computer on your local network):

- Port forwarding
- A localhost tunneling solution
- A virtual private network (VPN)

The implementation details are out of scope for this book, but I'll explain the principles behind these three ways and their strengths and weaknesses.

13.1.1 ● Port forwarding

The first way is the one you see advocated the most. It's not always easy to set up in

1 Except for your internet provider and a (dynamic) DNS provider.

practice, but it's conceptually simple and there's a lot of support for it in various software packages. With the free TLS certificates from Let's Encrypt it has become even easier in recent years.

In this approach you need to solve three issues:

- You need a domain name that always refers to your home's IP address, even if your internet service provider gives you a dynamic IP address.
- You need to forward a port from your router at home to your Raspberry Pi running your home automation software.
- You need a TLS certificate to have a secure connection to your home automation gateway.

If you have solved these three issues, you can point your web browser from anywhere in the world to your domain name, after which you get referred to your home automation gateway's web interface on an encrypted connection.

So let's look at these three issues one by one. Depending on your internet service provider, your modem at home gets a fixed IP address or a dynamic IP address. In the first case, you pay for a specific IP address and your modem always gets assigned this IP address as long as your contract is valid. In the second case, you just pay for a connection and you get assigned a random IP address, which can change any time.[2]

Whether it's fixed or dynamic, you're probably not able to remember your IP address, so you need to be able to address your home using a domain name. That's what DNS (Domain Name System) is for: it translates domain names to IP addresses.

If you have a fixed IP address at home, using a domain for your home is quite simple: you buy a domain name at a domain name registrar and you let it point to your IP address. The only thing you need to do after this is to pay the fee every year.

If you have a dynamic IP address, using a domain for your home becomes a bit more convoluted. You have to run software on your local network (on your router, server, or Raspberry Pi) that continuously checks your external IP address. After a change, the software sends your IP address to a dynamic DNS (DynDNS) service. This is a service where you have a subscription (there are many free ones) and that assigns you a subdomain of its domain.[3] Every time that your DynDNS software sends an updated IP address to the DynDNS provider, the subdomain refers to your new IP address. This way you can always refer to your home with your domain name. There are many options for DynDNS software, some of them are even included in routers as built-in software or can be easily installed

2 In practice, dynamic IP addresses don't change that often. But your contract doesn't guarantee that it stays the same, so you should really treat your IP address as one that could change any time.

3 I started using Duck DNS (https://www.duckdns.org) for dynamic DNS because Home Assistant has an easy-to-use add-on for it. Now I'm using Duck DNS with ddclient and I'm quite happy with this combination.

as add-ons. Later in this chapter, I show you how you do this with `ddclient` on your Raspberry Pi.

So now you have a way to refer to your home with a domain name, but this domain name refers to the IP address of your modem. Your home automation gateway doesn't run on your modem, but on a Raspberry Pi in your network. So if you're pointing your browser at your domain name, your modem gets a request to view its web pages. It will probably block this request because it's not a good idea to expose a modem's management interface to the world.

So how do you get this request from your web browser to your Raspberry Pi instead of to your modem? That's what port forwarding is for. Every time you connect to a web server with your web browser, you connect to port 80 for unencrypted HTTP or port 443 for encrypted HTTPS, or another one if you specify it in the URL. So what you want to do is this: when you connect to port 443 (or another one) on your modem's IP address, you want this request to be forwarded to port 443 (or another one, it doesn't even have to be the same as the original port) on your Raspberry Pi.

This mapping from one port on your modem's IP address to another (or the same) port on your Raspberry Pi's IP address is a setting you have to configure in your modem's web interface. The way to do this depends on your modem's model. Consult the documentation that your provider has given you, or consult the documentation for your router or its page on portforward.com (https://portforward.com/router.htm) if you have a router connected to the LAN port of your modem.

Warning:
This method for remote access requires that your Raspberry Pi has a fixed IP address on your local network. When it hasn't and the IP address suddenly changes, you can't reach your Raspberry Pi anymore from a remote location. Even worse, if by a stroke of bad luck another unsecured device on your network gets the IP address that your Raspberry Pi had, your port forwarding rule will make this other device widely available on the internet! So always configure your Raspberry Pi's local IP address to a fixed one, or add a static entry to the DHCP settings of your router. Consult your router's documentation for information about how to do this.

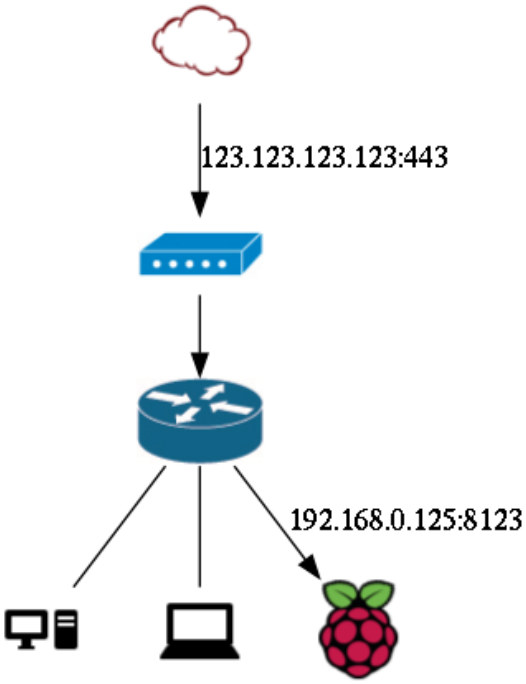

Figure 13.1 The router forwards incoming packets on port 443 of its external interface to port 8123 of the Raspberry Pi's IP address.

Now you can point your web browser to your domain name from anywhere, and you get access to your home automation gateway's web interface running on your Raspberry Pi. If you don't use HTTPS on your home automation gateway's web interface, this works now, but it's very insecure: everyone in the network path between you and your Raspberry Pi can eavesdrop on your communication.

The solution is to use HTTPS. But there's a problem: for HTTPS you need to have a valid TLS certificate for your domain. In the past, you had to purchase such a certificate, but for a few years, you have been able to get a free certificate from Let's Encrypt (https://letsencrypt.org), which is renewable in an automatic way. A lot of software supports Let's Encrypt. You need to run an ACME client (ACME stands for Automated Certificate Management Environment), that automatically requests and renews a TLS certificate for your domain. You can find a list of available ACME client implementations in the documentation of Let's Encrypt (https://letsencrypt.org/docs/client-options/).

So after you have a valid TLS certificate for your dynamic domain, when you point your web browser to your domain, you get referred to your home automation gateway's web interface and the communication is secured by TLS. This is one way for secure remote access to your home automation gateway.

Warning:
If you use this approach for remote access, everyone in the world can try to log into your home automation gateway.[4] So make sure that you're using a strong password (see Chapter 3). Moreover, one configuration mistake (for instance disabling or resetting the password, or allowing registration of new users) can make your home automation gateway wide open for everyone. That's why I don't advocate this approach.

13.1.2 ● A localhost tunneling solution

Another solution is localhost tunneling. On your home automation gateway, you create a network tunnel that leads to a relay server on the internet. Now when you (or anyone else) visits the relay server in a web browser, all traffic is forwarded through the tunnel to your home automation gateway at home.

With this method, you don't need to set up dynamic DNS or port forwarding at home, because the tunnel is initiated at your side. As soon as the tunnel is running, there's two-way communication between your home automation gateway and the relay server. So this could be an interesting solution if your modem is not flexible enough to set up dynamic DNS or port forwarding.

There are a lot of services that offer a central relay server for localhost tunneling. As part of your account (free or paid), you receive one or more subdomains of the service's domain. At first sight, this subdomain looks like a dynamic DNS domain. The difference is subtle but important: your subdomain at the localhost tunneling service doesn't point to your home IP address, but to the relay server. So no one except the localhost tunneling service knows your IP address. Many users find this thought comforting.

Two popular examples of these services are:

Pagekite (https://pagekite.net)
You get your own subdomain https://yourname.pagekite.me. The service is free for individuals (or a suggested donation of $ 3 per month). The software is completely open-source and you can even use it to create your own relay server, for example on a publicly accessible VPS (virtual private server) that you rent.

Ngrok (https://ngrok.com)
It has a free plan with a random subdomain, and for $ 5 per month, you get custom subdomains. If you want an end-to-end TLS tunnel, you have to pay more. It's not open-

4 You can prevent this by introducting TLS certificates for the client. This way the client not only authenticates the server by its server certificate, but the server (your home automation gateway) also authenticates your client by its client certificate. If a client tries to connect to your home automation gateway and can't offer a trusted certificate, the connection is declined and it can't even try a password. However, using TLS client certificates adds some complexity and it hasn't really caught on.

source, and you can't run an ngrok server on your own VPS.[5] It has a few nice features for web development, such as inspection of HTTP requests and traffic. For secure remote access to your home automation gateway, I don't recommend it.

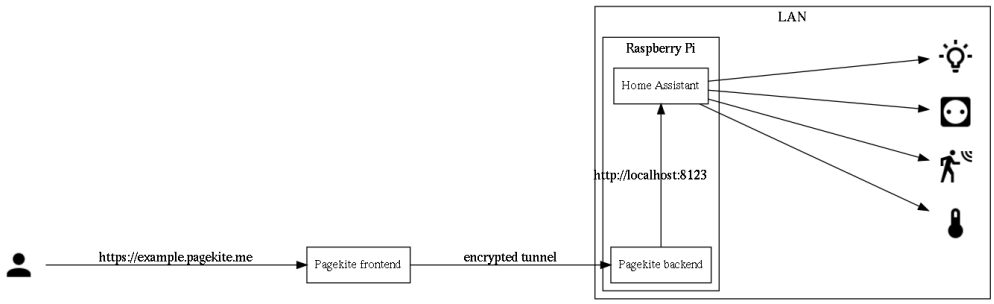

Figure 13.2 A localhost tunneling solution such as Pagekite creates an encrypted tunnel to a web server running in your home automation gateway.

Note that you should not just use one of these services without thinking thoroughly about what they are offering exactly. I already mentioned that with ngrok you have to pay for an end-to-end TLS tunnel. What does this mean? There are two ways in which these services offer their tunneling solution. I'll use Pagekite as an example:

With a certificate of the service
If you get a subdomain https://yourname.pagekite.me and you inspect its certificate in your web browser, you'll see that it's a wildcard certificate for `*.pagekite.me`. This means that all users of Pagekite share the same certificate. You're not in control of the private key of this certificate: the company behind the Pagekite service is. So the connection between you and the Pagekite service is encrypted, and the connection between the Pagekite service and your home automation gateway is encrypted, but in principle, all Pagekite employees can intercept your traffic.

With your own certificate, end-to-end encrypted
If you create your own certificate, only you have access to the certificate's private key. So you can run your home automation gateway at home using this certificate, and all communication between you and your home automation gateway is end-to-end encrypted. Even Pagekite employees can't eavesdrop on your communication: they only see encrypted network traffic. To use your certificate, you have to pay a domain name registrar for your domain and then get a certificate for it, for example with Let's Encrypt, or use your own certificate authority and add its certificate to all your client devices. This is more difficult to set up, see Chapter 3.

All in all, a localhost tunneling solution is a bit easier to set up than using dynamic DNS and port forwarding, but for the most secure way, you still need your own TLS certificate.

5 The product page mentions that you can license a dedicated installation of the ngrok server cluster for commercial use on the Amazon Web Services cloud, but it's completely managed by the ngrok company, you even have to give them the keys to your AWS instance.

The same warning applies here: everyone in the world can try to log in to your home automation gateway. That's why I also don't advocate this way.

Note:
There's another way that looks like the localhost tunneling solution: SSH remote port forwarding. With this method, you create a tunnel with an `ssh` command on your home automation gateway to a publicly accessible server such as your VPS. As long as the tunnel is open, network traffic to the specified port on your VPS will be tunneled in an encrypted (SSH) connection to your home automation gateway at home. While this is secure and easy to set up, it's not the most efficient way and it's more suited for temporary access. If you want to know more, read `man ssh` and search for the `-R` option.

13.1.3 ● A virtual private network (VPN)

Both plain port forwarding and a local tunneling solution have some security measures but mostly trust that your home automation gateway they point to is secured. Moreover, a configuration mistake can wreak havoc on your network.

A better solution is a virtual private network (VPN). If you make a configuration mistake there, chances are higher that your connection to the home automation gateway doesn't even work then.

So the approach with a VPN becomes this. Anywhere in the world, you can connect to your modem at home. This requires you to have a domain name pointing to your IP address, and thus probably a dynamic DNS. You also need to know the port number of your VPN server. By default, this is UDP port 1194 for OpenVPN, and 51820 for WireGuard. But you don't connect to the VPN with your web browser: you have to connect to it with a VPN client. Your VPN server running at home authenticates your VPN client using a certificate, a long key, password, or any other means.[6]

Only when you can authenticate to your VPN server, the VPN connection is set up. This creates a virtual network that links you to your network at home, with all traffic between your VPN client and the VPN server encrypted. As long as the VPN connection is open, you can access your home automation gateway as if you're on your home network.

There are many ways to work with a VPN at home. First, there's the choice of where you want to run the VPN server:

On a combined modem/router
If you have a combined modem/router at home and you can install a VPN server on it, you don't even need port forwarding: you just have to set up the VPN server and allow traffic to come into its designated port.

6 You can even authenticate your VPN client using a hardware key such as a FIDO U2F security key.

On a router behind your modem
If you have a separate router connected to the LAN port of your modem, you can install a VPN server on it. You only have to forward the port of your VPN server software on your modem so the traffic on this port goes to your router running the VPN server.

On a Raspberry Pi
If you can't install a VPN server on your router, you can always install it on a Raspberry Pi on your local network, even on the same Raspberry Pi as the one running your home automation gateway. You need to forward the port of your VPN server software on your combined modem/router to your Raspberry Pi. If you have separate modem and router devices, you need to forward the port on your modem to your router, and on your router to your Raspberry Pi.

As soon as you're connected to the VPN from outside, you have access to all your devices on your home network, including your home automation gateway.

The second choice you have to make is which VPN server software. Two popular choices are:

OpenVPN (https://openvpn.net)
The community version of OpenVPN is open-source. It's using TLS to create a secure tunnel, is very flexible, and extensible can be installed on many operating systems and devices, including OpenWrt routers. The downside is that it's a quite complex codebase, and complexity is the enemy of security.

WireGuard (https://www.wireguard.com)
This is a relatively new VPN protocol (created in 2015), but it has already made quite a splash. It's open-source, easy to set up and use, highly performant, and has a small codebase, partly because it doesn't use OpenSSL (in contrast to OpenVPN). It's using state-of-the-art cryptography and the protocol is formally verified. There's one big downside: if a fundamental weakness is found in one of the protocols it uses, the WireGuard protocol can't be fixed without breaking compatibility, because the protocol is not extensible.

Because I value security higher than extensibility and I don't trust complex software, my preference is WireGuard. This is what I'll show further in this chapter: how to install WireGuard on your Raspberry Pi running your home automation gateway, and how you connect to it on your smartphone or laptop when you're on the go.

Note:
You can also run WireGuard on your router, especially if it's running an open-source operating system such as OpenWrt or OPNsense. I prefer this way. A lot of readers don't have this option, and I want all readers to have a way to secure remote access to their home automation gateway. WireGuard on a Raspberry Pi offers this.

13.2 ● Updating your dynamic DNS with ddclient

If you want to connect to your VPN server at home and you don't have a fixed IP address for your internet connection, you first need a dynamic DNS service. Many of these services offer their domains for free. Some examples are No-IP (https://www.noip.com), FreeDNS (https://freedns.afraid.org), and Duck DNS (https://www.duckdns.org). Register a domain with one of these services. In the rest of this section, I assume you have registered a subdomain of Duck DNS, example.duckdns.org.

As I already told you, you could run a dynamic DNS updater on your router, but your Raspberry Pi can do it too. The software you use for it is called ddclient (https://ddclient.net). Installing it directly on Raspberry Pi OS is easy, but the LinuxServer people (https://www.linuxserver.io) have created a Docker container, which makes it even easier to integrate the dynamic DNS update with the architecture advocated in this book based on a Docker Compose file.

First create a directory for the ddclient container to store its configuration file in:

```
mkdir -p /home/pi/containers/ddclient
```

Then edit your docker-compose.yml file in your home directory so it has the following content:

```
version: '3.7'

services:
  # other containers
  ddclient:
    image: linuxserver/ddclient
    container_name: ddclient
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Brussels
    volumes:
      - ./containers/ddclient:/config
    restart: always
```

Change the `TZ` environment variable to your location's time zone.

Now create a configuration file for ddclient:

```
nano /home/pi/containers/ddclient/ddclient.conf
```

Have a look at the list of supported dynamic DNS protocols (https://sourceforge.net/p/ddclient/wiki/protocols/) for the right syntax for your dynamic DNS provider.

For Duck DNS, the file should look like this:

```
ssl=yes
use=web
protocol=duckdns
password=TOKEN
example.duckdns.org
```

Replace `TOKEN` by your API token for Duck DNS, and change the domain on the last line of the configuration file to your subdomain registered at the dynamic DNS service. Save the file, exit nano, and then start the container:

```
docker-compose up -d
```

Wait for a while until the container is started, and then look at the logs:

```
docker logs -f ddclient
```

On the list line, you should see **SUCCESS** and the message that your dynamic DNS is linked to your IP address. Compare this IP address to the one you see in https://www.whatismyip.com: it should be the same.

From now on, ddclient periodically checks for a change of your IP address and then updates your dynamic DNS.

13.3 • Running WireGuard on your Raspberry Pi

Before you start installing WireGuard, make sure Raspberry Pi OS is up-to-date and especially that it has installed the newest kernel version. After this, reboot to use the updated kernel:

```
sudo apt update
sudo apt upgrade
sudo reboot
```

This update is needed because WireGuard is a kernel module and the version you install should not be newer than the running kernel version.

Because WireGuard is tightly coupled to the Linux kernel, I don't install it in a Docker container, but directly on Raspberry Pi OS.

There are still two things you should take care of in your network:

Static IP address
Give your Raspberry Pi a static IP address. See Chapter 2 for more details.

Port forwarding
Forward UDP port 51820 from your modem/router to the IP address of your Raspberry Pi. After this, you can install WireGuard.

13.3.1 ● Installing PiVPN

You could install and configure WireGuard manually, but there's an interesting tool that makes this much more user-friendly: PiVPN (https://www.pivpn.io).

PiVPN can be installed like this:

```
git clone https://github.com/pivpn/pivpn.git
sudo bash pivpn/auto_install/install.sh
```

The install script checks whether you have enough free storage space, and it installs everything that's needed. It lets you choose the network interface (**eth0** for the Ethernet interface) and whether you want to use DHCP or a static address. Choose DHCP if you have set up a static mapping in your router's DHCP settings.

Confirm that you want to store your VPN settings in the `pi` user's home directory. Then choose WireGuard as the VPN protocol (OpenVPN is the other choice). Confirm the default port (51820) and then choose the DNS server for the VPN clients. By default, Quad9 is selected. If you have a DNS server running on your home network (for instance on your router), use **Custom** and enter its IP address.

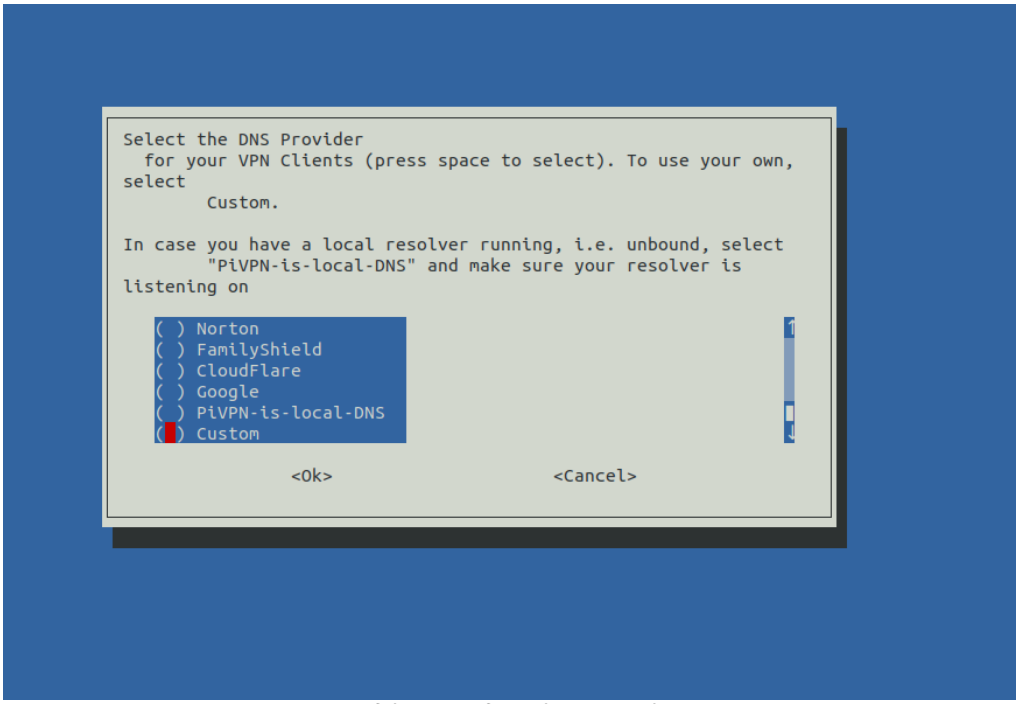

Figure 13.3 Use one of the preconfigured DNS providers or use your own.

In the next step, the install script enters the public IP address of your internet connection. If your internet provider doesn't assign you a fixed IP address, choose the option **DNS Entry** and enter the dynamic DNS domain that you keep updated with ddclient.

After this, the install script generates VPN keys. At the end, it asks you to reboot.

13.3.2 ● Adding a VPN client

Now your WireGuard VPN is ready, and you only have to add a client. PiVPN makes this very easy:

```
pivpn add
```

Enter a name for the client. After this, PiVPN generates a key pair and a configuration file for the client, adds the client to the WireGuard server configuration, and restarts the WireGuard server on your Raspberry Pi.

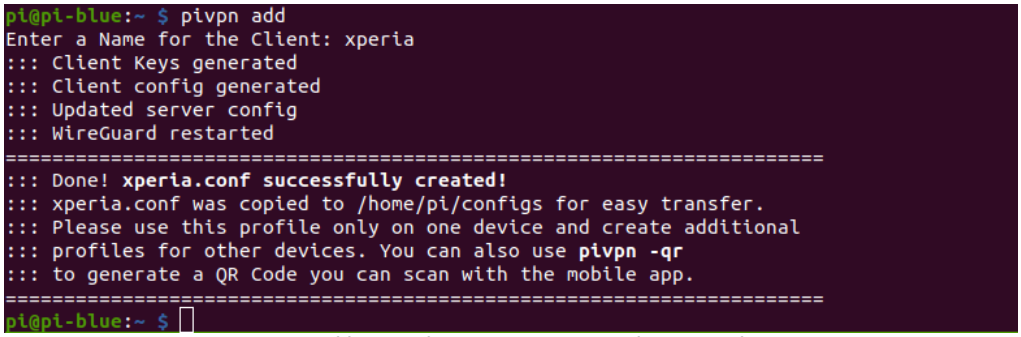

Figure 13.4 Add a VPN client to your WireGuard server with PiVPN.

Now you have to get this VPN configuration on your client device that you want to connect to your Raspberry Pi's VPN from outside your network. On an Android or Apple smartphone you can run the official WireGuard app. Then enter the following command on your Raspberry Pi:

```
pivpn qrcode
```

Enter the name of your client. After this, you see a QR code that you can scan on your smartphone. In the WireGuard app, press the blue plus sign at the bottom right and choose **Create from QR code**. Scan the QR code on your screen, give the VPN tunnel a name, and choose **Create tunnel**.

Figure 13.5 The QR code is an easy way to import your VPN configuration into the WireGuard app on your smartphone.

13.3.3 ● Connecting with a VPN client

Now the WireGuard app on your smartphone has your VPN configuration with the secret key that is needed to connect to your VPN server. You only have to enable the switch in your app next to the name you assigned to the tunnel, and you're connected.

> **Note:**
> Make sure you're not connected to your home Wi-Fi network when you're testing the VPN connection. Disable Wi-Fi on your smartphone and connect from your mobile data connection.

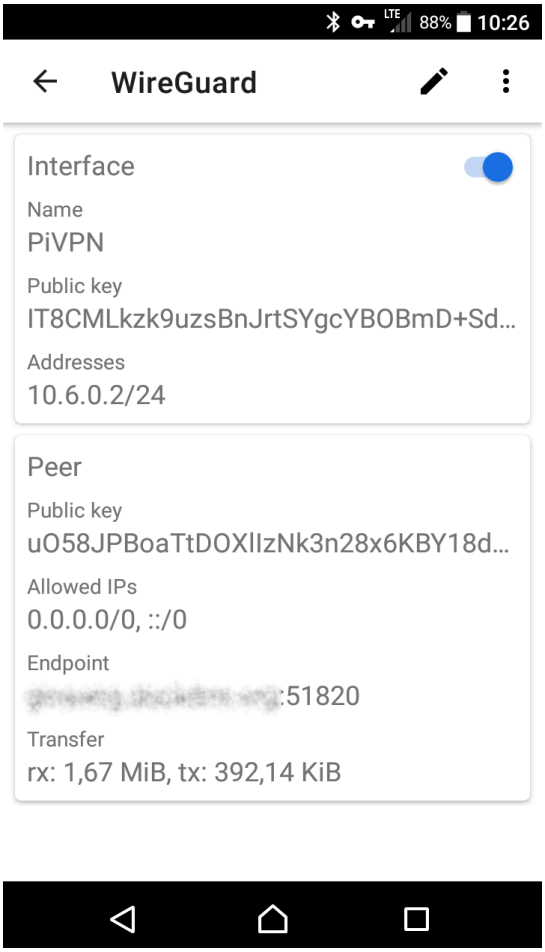

Figure 13.6 The WireGuard app for Android lets you connect to your Raspberry Pi at home.

When you're connected to the VPN, you are now able to access your home automation gateway by surfing to the IP address of your Raspberry Pi on your local network. This lets you access the web interfaces of Zwave2mqtt, Home Assistant, Node-RED, and so on. You are also able to access other devices on your local network, which is interesting if you have spread your home automation installation over multiple devices.

Note:
Your VPN clients use the DNS server you have configured while installing PiVPN. If you entered the IP address of your internal DNS server which is running on your router, you can access your gateway by its internal hostname, such as raspberrypi.home. This is required if you want to access your services over HTTPS.

PiVPN does more than you have asked for: it has installed firewall and packet forwarding rules so you can even surf on the internet from your smartphone using your internet connection at home while you're abroad. This is interesting to be able to surf the internet over an encrypted tunnel while you're on an insecure network, but if you won't use this, you can disable it. Just open the configuration file for system settings:

```
sudo nano /etc/sysctl.conf
```

Search for this line:

```
net.ipv4.ip_forward=1
```

And put a hash sign (`#`) at the beginning of that line to disable packet forwarding. Reboot and check your VPN connection again. This time you can only connect to devices on your local network when you have your smartphone connected to your VPN.

Note:
If you encounter trouble with PiVPN, have a look at the wiki where common problems and their solutions are listed (https://github.com/pivpn/pivpn/wiki/FAQ). The first thing you can do is run the `pivpn debug` command, which does a self-test and shows a lot of information about your configuration. If you don't find a solution this way, open an issue on GitHub (`https://github.com/pivpn/pivpn/issues`) and show the output of `pivpn debug`.

13.3.4 ● Managing your VPN clients

PiVPN offers a couple of simple commands to manage your VPN clients. If you want to get a list of all clients you have defined, just enter:

```
pivpn list
```

For every client, you see its name, and public key when you have created the configuration.

If you want to know which clients are currently connected to the VPN, enter:

```
pivpn clients
```

This shows you a table with clients with their name, external IP address, virtual IP address on the VPN, how many bytes have been sent and received by the client and when the client was last active.

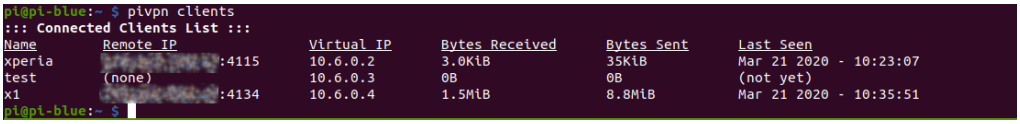

Figure 13.7 PiVPN shows you a list of connected VPN clients to your WireGuard server.

There's another command to remove a client:

```
pivpn remove
```

Choose the name of the client. After this, the WireGuard server doesn't allow the public key of your client anymore, so the client can't connect to your VPN.

Warning:
If you have lost your smartphone, immediately remove its configuration from your Raspberry Pi with pivpn remove to prevent someone from having access to your network!

13.4 ● Summary and further exploration

In this chapter, you learned how to access your home automation gateway securely from outside your home. I showed you three ways for remote access, and I discussed their strengths and weaknesses. You learned how to update your dynamic DNS with ddclient and how to run the WireGuard VPN service on your Raspberry Pi with PiVPN.

In the spirit of self-hosting, this makes it possible to remotely get access to your home automation system securely, with the least reliance on other parties as possible.

If you don't like the VPN approach, feel free to try the other ways for remote access. If you do like VPN, I recommend you to explore WireGuard more, and perhaps try to install it yourself without PiVPN. It gives you more control over the configuration. There's also experimental work to better secure your WireGuard connections with two-factor authentication.

Chapter 14 • Conclusion

In this book, I have shown you step by step that it's possible to create a fully self-hosted home automation system and that you can even do this on a Raspberry Pi, including advanced functionality such as a voice assistant.

Let's reiterate the four properties of a good home automation system that I introduced in Chapter 1. If you have followed the instructions in this book closely, your home automation system is now:

Secure
You make as much use as possible of encryption and authentication, updated software, and isolated services in Docker containers. This minimizes the risk if one of the components of your home automation system becomes vulnerable.

Modular
You can add other devices and even other home automation protocols, because your home automation system has an extensible architecture, using MQTT to communicate between various services.

Open-source
If you encounter problems, you can delve into the code of all software used in your home automation system. You can fix errors yourself and if every user does that, your home automation system keeps getting better and better.

Self-sufficient
Your home automation system keeps working when your internet connection is down, and you don't depend on any third-party services that render your home automation system useless when they stop. You are in full control of your home, as it should be.

To wrap up this book, I'll show you one more service, to give you an overview of all your home automation services, and I give some pointers to other interesting projects that I couldn't cover in this book.

14.1 • A dashboard for all your services

Let's install one more service: Heimdall (https://heimdall.site), which is an application dashboard. By now you have various services with a web interface running, but do you remember which port you have to use for each one of them? Of course not, and that's where Heimdall comes to help: it lets you define a dashboard with an icon for each of your services, so you only have to visit Heimdall's page on your Raspberry Pi and there you go: all your home automation services are just a click away.

Create a directory for Heimdall and copy the TLS key and certificate of your Raspberry Pi to the locations where Heimdall expects them:

```
mkdir -p /home/pi/containers/heimdall/keys
cp /home/pi/containers/certificates/key.pem
/home/pi/containers/heimdall/keys/cert.key
cp /home/pi/containers/certificates/cert.pem
/home/pi/containers/heimdall/keys/cert.crt
```

And then add the following container definition to your `docker-compose.yml` file:

```
version: '3.7'

services:
  # other services
  heimdall:
    image: linuxserver/heimdall
    container_name: heimdall
    restart: always
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Brussels
    volumes:
      - ./containers/heimdall:/config
    ports:
      - 443:443
```

And start the container:

```
docker-compose up -d
```

In the logs (`docker logs -f heimdall`), you should see the message "using keys found in /config/keys". After this, a new Heimdall configuration is installed in the container, and this takes a while. When you see the message "Creating app key. This may take a while on slower systems", expect to wait a couple of minutes on a Raspberry Pi 4, longer on a less powerful model.

When the container has started, you can just visit https://HOSTNAME in your web browser, with HOSTNAME the hostname of your Raspberry Pi. This will show you an empty dashboard with the message "There are currently no pinned applications, Add an application here or Pin an item to the dash".

First set a password on the admin user, because otherwise, everyone in your network has access to your dashboard. Click on the user icon at the bottom right of the page and then

on the edit button next to the admin user. Enter a name and email address for the user, and choose and confirm a password. You can even upload an image file as an avatar. Click on **Save** to change the user.

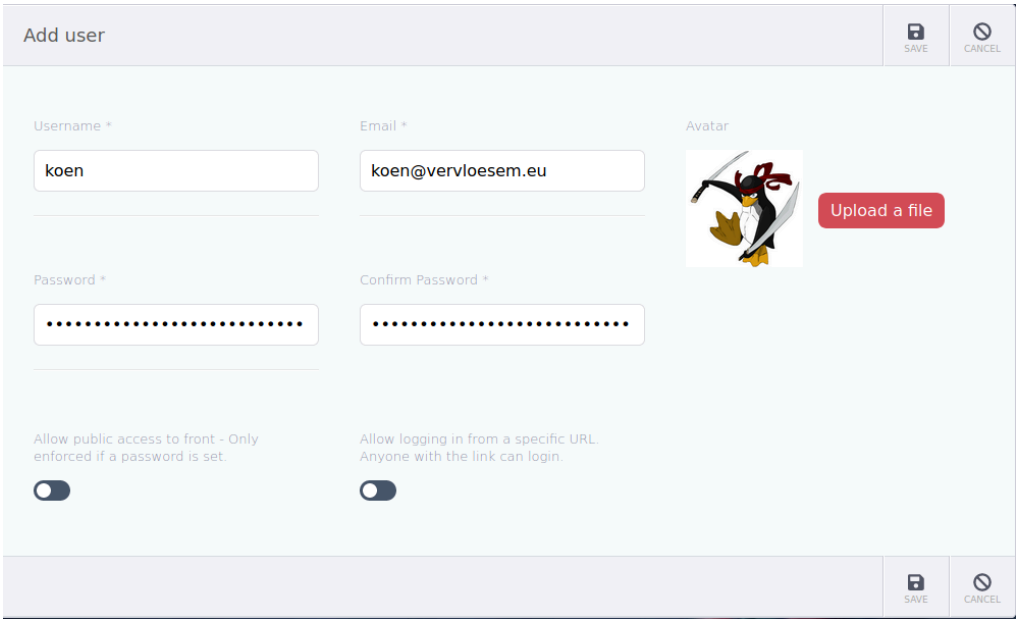

Figure 14.1 You can secure Heimdall's dashboard with a password for each user.

Now click on the items icon (just below the user icon) and then on **Add**. Enter the name of your service. If the service is in the list of supported application types (such as Gotify, Home Assistant, and Node-RED), this will automatically fill in an icon, application type, and color. Otherwise, upload an icon and/or choose a color (**#161b1f** is a nice one for the default background). Also, enter its URL with the right port number on your Raspberry Pi. Don't forget to enable **Pinned** at the top and then click **Save**. After this, there's an icon for your service on the dashboard. When you click on it, a new tab opens with the web page for the service.

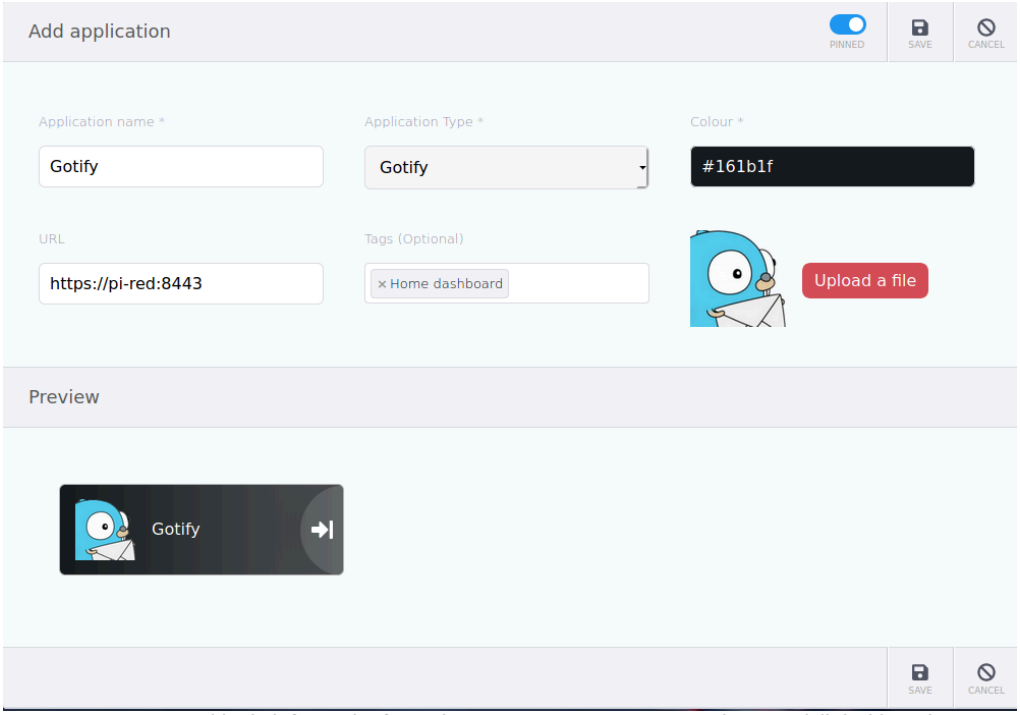

Figure 14.2 Add a link for each of your home automation services to the Heimdall dashboard.

If you now add all your services this way to Heimdall's dashboard, you have them all in one place and you don't have to remember their port names anymore. If anyone on the network wants to access the dashboard, they are asked first to choose a user and enter a password.

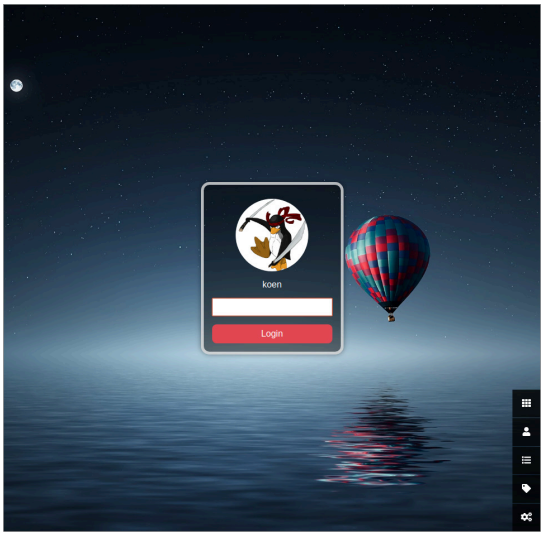

Figure 14.3 Log in to your Heimdall dashboard to get access to all your home automation services.

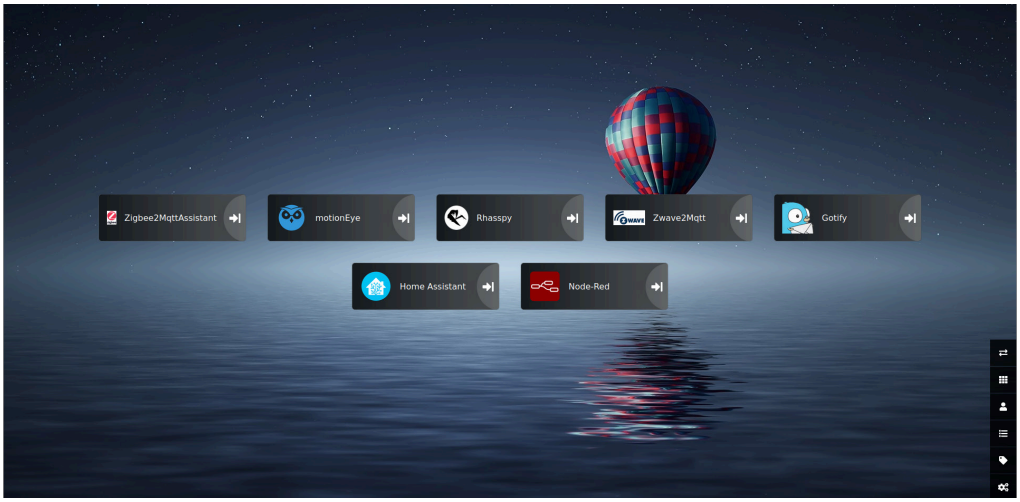

Figure 14.4 A Heimdall dashboard with all the services from this book that have a web interface.

14.2 • More about home automation

In this book, I covered a selection of home automation services, but like every book with such a broad topic, there are services I could only shortly explain and other services that I couldn't cover at all. To make up for the latter, here are some interesting projects that I recommend you to explore.

Jan-Piet Mens, the creator of mqttwarn (see Chapter 11), has developed another interesting project, OwnTracks (https://owntracks.org). This is an open-source project that lets you keep track of your location. You install the server on your Raspberry Pi and run an app on your Android or iOS phone, that continuously updates your location to the server using MQTT. This way you can automate things like turning up the heat in winter when you're almost home or sending you a notification when your spouse leaves at work, all without having to send your location to an untrusted third party.

I also haven't covered other AI (artificial intelligence) topics than voice control. For instance, you could add realtime object detection to your motionEye set up with a system such as Frigate (https://github.com/blakeblackshear/frigate). This analyzes images from an RTSP camera on your network with OpenCV and TensorFlow Lite, and it tells you whether it sees a person, a dog, or something else. Frigate can send this information to an MQTT broker, so it perfectly fits in this book's architecture. However, the processing power needed for object detection is big: you need a Google Coral USB Accelerator and preferably a Raspberry Pi 4 to fully use the USB 3 throughput of the accelerator dongle.

Concerning wireless technologies, you could also add infrared support. With an infrared transmitter on your Raspberry Pi you could let it control your TV or music installation, and with an infrared receiver connected you could control your home automation gateway with an infrared remote. An infrared transmitter or receiver is quite cheap and can be easily

connected to the Raspberry Pi's GPIO pins. The software to control all this is LIRC, which stands for Linux Infrared Remote Control (https://www.lirc.org).

If you want a longer range, LoRa (Long Range) is an interesting technology to explore. It is a low-power wide-area network (LPWAN) protocol, which can reach distances of a few kilometers. LoRA uses the 868 MHz or 433 MHz frequency band in Europe, 915 MHz in North America and Australia, and 923 MHz in Asia. The LoRA protocol is quite resilient against interference, which is important because it shares its frequencies with other protocols (such as Z-Wave in Europe). It's a perfect technology if you want to have temperature and humidity sensors in a big garden. You only have to add a LoRA HAT expansion board on a Raspberry Pi to communicate with your LoRA devices.

This is the end of this book, but it's surely not the end of your (and my) home automation adventure. You can keep tinkering with your setup, adding new functionality, automating more things, and so on. Home automation is an addictive hobby. Like with many hobbies, it's not the destination that matters but the journey, and meanwhile, you learn about a lot of fascinating topics and powerful technology. I can't imagine a better companion for this journey than the Raspberry Pi.

• Appendix

This appendix lists some specialized tips that could come in handy in various situations while working on home automation with your Raspberry Pi:

- Getting the name and ID of a serial device
- Switching USB ports on and off
- Disabling onboard Bluetooth and/or Wi-Fi
- Disabling the on-board LEDs from the Raspberry Pi or the Camera Module
- Securing insecure services with a reverse proxy
- Bridging two MQTT brokers securely

15.1 • Getting the name and ID of a serial device

If you have connected a device to your Raspberry Pi and you want to use it, you have to determine the device name that Raspberry Pi OS has assigned to it. For serial devices (such as a Z-Wave or Zigbee device) you should see the right path for your setup if you execute the command `dmesg | grep tty` after connecting the device. For instance, if you see `ttyACM0` as the device name, this makes the device path `/dev/ttyACM0`.

However, if you're applying a multi-protocol approach as advocated in this book, and thus you have connected multiple serial devices, you shouldn't rely on these device names, as the name a device gets assigned depends on the order of loading the device driver. It's perfectly possible that your Z-Wave and Zigbee transceiver swap device names after a reboot or upgrade of the Linux kernel.

That's why you should use the device name by ID. You can find these in the output of `ls -l /dev/serial/by-id`.

For the CC2531 Zigbee transceiver, you should see a name such as `usb-Texas_Instruments_TI_CC2531_USB_CDC___0X00124B0018E31EDB-if00`. This means that you'll use the device `/dev/serial/by-id/usb-Texas_Instruments_TI_CC2531_USB_CDC___0X00124B0018E31EDB-if00`.

For Aeotec's Z-Stick, you should see a name such as `usb-Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001-if00-port0`, which makes the device path `/dev/serial/by-id/usb-Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001-if00-port0`.

```
pi@pi-red:~ $ ls -l /dev/serial/by-id
total 0
lrwxrwxrwx 1 root root 13 Feb 16 11:17 usb-Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001-if00-port0 -> ../../ttyUSB0
lrwxrwxrwx 1 root root 13 Mar  1 01:27 usb-Texas_Instruments_TI_CC2531_USB_CDC___0X00124B0018E31EDB-if00 -> ../../ttyACM0
pi@pi-red:~ $ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 Feb 16 11:17 /dev/ttyUSB0
pi@pi-red:~ $ ls -l /dev/ttyACM0
crw-rw---- 1 root dialout 166, 0 Mar  1 01:27 /dev/ttyACM0
pi@pi-red:~ $
```

Figure 15.1 If you have multiple serial devices attached, you better use the device ID.

15.2 ● Switching USB ports

In various situations it could be interesting to shut off power to specific USB ports, temporarily or for a longer time:

- Your Z-Wave transceiver is stuck in a non-working state. Shutting down the USB port and immediately re-enabling it resets the transceiver so it works again. You can do this without having to walk to the Raspberry Pi, pulling the transceiver out and re-inserting it. Two commands on a remote SSH session are enough.
- You have connected a USB-powered device such as a fan to your Raspberry Pi that you want to easily enable and disable. You can do this without having to fiddle with GPIO pins or complex home automation protocols.

On some USB hubs, you can control USB power on individual ports. Only very few hubs support per-port power switching.

Two projects let you control USB ports:

- hub-ctrl.c (https://github.com/codazoda/hub-ctrl.c)
- uhubctl (https://github.com/mvp/uhubctl)

The second project is actually in the Raspberry Pi OS repository, but you might want to install it from source to get a newer version with support for the Raspberry Pi 4's USB ports.

Both projects can be used to control the internal USB hub of the Raspberry Pi, but this only works for all USB ports at the same time, not for individual USB ports. This is a limitation of the Raspberry Pi hardware design.

Note:
As a workaround, you can buy an external USB hub, attach it to any USB port of the Raspberry Pi, and connect the USB devices you want to control individually to this USB hub.

15.3 ● Disabling the onboard radio chips

In various situations, you'd like to disable the on-board Bluetooth and/or Wi-Fi chips. For instance:

- You want to use the RaZberry daughterboard for Z-Wave on your Raspberry Pi, but the on-board Bluetooth chip already uses the UART it needs.
- You want to eliminate interference between the on-board Bluetooth and/or Wi-Fi radio and your Zigbee transceiver, which all operate in the crowded 2.4 GHz frequency band.
- You want to save as much power as possible and you don't need the onboard Bluetooth and/or Wi-Fi chips.
- You want to use an external USB adapter for Bluetooth and/or Wi-Fi with a better

range.

Luckily the on-board radio chips can be easily disabled.

15.3.1 ● Disabling onboard Bluetooth

You can disable the on-board Bluetooth chip with the following line in `/boot/config.txt`:

```
dtoverlay=disable-bt
```

This will free the UART so it can be used by other devices such as the RaZberry and restores the UART function to GPIOs BCM14 and BCM15 (physical pins 8 and 10 on the GPIO header).

Next, disable the system service that initializes the modem so it doesn't use the UART anymore:

```
sudo systemctl disable hciuart
```

Then reboot your Raspberry Pi to check whether the Bluetooth device has been disabled: `hciconfig -a` shouldn't show the on-board Bluetooth device anymore.

15.3.2 ● Disabling onboard Wi-Fi

You can disable the onboard Wi-Fi chip with the following line in `/boot/config.txt`:

```
dtoverlay=disable-wifi
```

Then reboot your Raspberry Pi to check whether the Wi-Fi device has been disabled: `ip link show` shouldn't show the `wlan0` device anymore.

15.4 ● Disabling the on-board LEDs

In some situations you'd like to disable the Raspberry Pi's on-board LEDs:

- You want your Raspberry Pi Zero (W) to use as little power as possible, maybe because it's powered by a battery or solar panel.
- You're annoyed by the glaring red LED on your Raspberry Pi with the Camera Module that's watching your hallway at night.

Luckily the LEDs on various Raspberry Pi models can be enabled and disabled individually.

15.4.1 ● Raspberry Pi Zero (W)

The Raspberry Pi Zero (W) has only one LED. By default, it's on when it has power, and it temporarily goes off for activity on the microSD card. So whenever your little Pi is active, you see the LED flashing.

To disable the LED completely, add the following lines to the `/boot/config.txt` file:

```
dtparam=act_led_trigger=none
dtparam=act_led_activelow=on
```

After a reboot, the LED stays off.

15.4.2 ● The big Raspberry Pi models

The bigger Raspberry Pi models (including the Raspberry Pi 3A+) show the board's activity on two LEDs:

POW (red)
The board gets enough power.

ACT (green)
The microSD card is active.

You can disable these LEDs on all these models except the original Raspberry Pi Model B with the following lines in `/boot/config.txt`:

```
dtparam=pwr_led_trigger=none
dtparam=pwr_led_activelow=off

dtparam=act_led_trigger=none
dtparam=act_led_activelow=off
```

After rebooting, the LEDs stay off.

The green ACT led isn't as annoying as the red POW led. So if you just want to disable the PWR LED, ignore the last two lines.

15.4.3 ● Ethernet models

The Raspberry Pi models with an Ethernet interface have two other LEDs for the network interface:

LNK (green)
On when a link is established, flashes when there's network activity on the Ethernet interface.

SPD (yellow)
On when there's a 100 Mbps connection or higher.

On the original Raspberry Pi Model B, these LEDs were positioned next to the POW and ACT LEDs; the newer models have them on the sides of the Ethernet socket.

On the Raspberry Pi 4B you can disable both LEDs with the following lines in `/boot/config.txt`:[1]

```
dtparam=eth_led0=4
dtparam=eth_led1=4
```

On the Raspberry Pi 3B+ the lines should be:

```
dtparam=eth_led0=14
dtparam=eth_led1=14
```

On older models of the Raspberry Pi you need the software LAN951x LED control (https://mockmoon-cybernetics.ch/computer/raspberry-pi/lan951x-led-ctl/):

```
lan951x-led-ctl --spd=0 --lnk=0
```

This disables both the SPD and the LNK LEDs. You need to run this program after each boot of the Raspberry Pi, for instance by putting the command in the startup `script /etc/rc.local`.

15.4.4 ● Raspberry Pi Camera Module

The Raspberry Pi Camera Module has its own LED which lights up when the camera is recording. You can disable it by adding the following line to `/boot/config.txt`:

1 You can find all the possible values in https://github.com/raspberrypi/firmware/blob/master/boot/overlays/README.

```
disable_camera_led=1
```

Note:
Some other CSI cameras for Raspberry Pi use this setting for something else, for instance, to enable night vision mode for cameras with infrared lighting.

15.5 ● Securing insecure web services with a reverse proxy

Some projects in this book don't offer an encrypted connection for their web interface. However, this doesn't mean that you have to use them unsecured. You can create a reverse proxy. This software:

- listens on port X of your Raspberry Pi for an HTTPS connection;
- forwards all traffic unencrypted to port Y of the service.

This all happens transparently for the user: you can just use HTTPS, you can verify the TLS certificate and there's nothing that tells you that the underlying service is using HTTP. One such reverse proxy is Nginx (https://nginx.org). You can install it in a Docker container, let it listen on port X of Your Raspberry Pi for an HTTPS connection, and then forward traffic unencrypted over Docker's internal network to port Y of another Docker container. As the unencrypted connection stays on your Raspberry Pi, you have secured your connection.

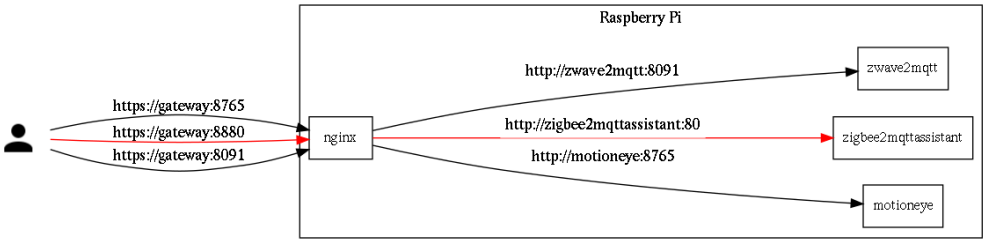

Figure 15.2 With nginx working as a reverse proxy for the other containers, you can use HTTPS to access services that only support HTTP.

Moreover, you can add authentication in your reverse proxy too if the other service doesn't support it. When connecting to the reverse proxy, you have to supply a username and password, and only if they are correct, you are forwarded to the service.

15.5.1 ● Using nginx as a reverse proxy with HTTPS

As an example, let's install and configure nginx as a reverse proxy for Zwave2Mqtt using TLS.

First create a directory for your nginx container and enter it:

```
mkdir /home/pi/containers/nginx
cd !$
```

Create Diffie-Hellman parameters for the TLS connection:

```
openssl dhparam -out dhparams.pem 2048
```

Note:
This command takes a couple of minutes, even on a Raspberry Pi 4.

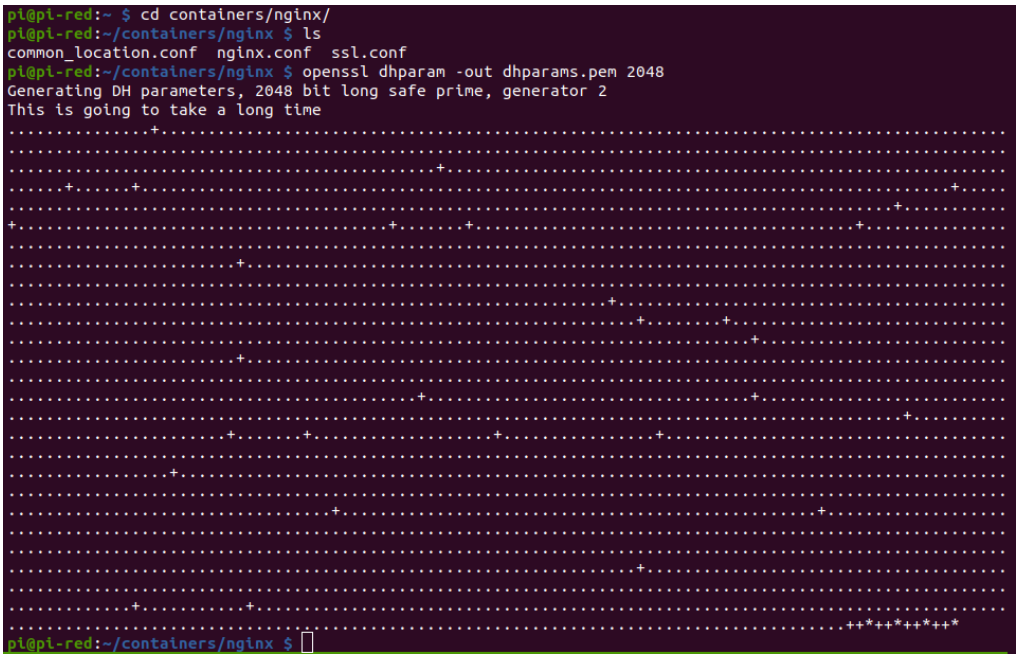

Figure 15.3 Creating Diffie-Hellman parameters takes a long time on a Raspberry Pi.

Then you need a configuration file for nginx. Because you can use nginx as a reverse proxy for more than one other service, it's recommended to split the configuration file for easier reuse.

First create a configuration file `common_location.conf` with this content:

```
proxy_set_header    X-Real-IP           $remote_addr;
proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
proxy_set_header    X-Forwarded-Proto   $scheme;
proxy_set_header    X-Forwarded-Host    $host;
proxy_set_header    X-Forwarded-Port    $server_port;
proxy_set_header    Host                $host;
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "Upgrade";
```

This just contains some HTTP headers that the reverse proxy sets. Not all of them are strictly needed for all services, but for instance, the connection upgrade headers are needed for the WebSocket protocol to work.

Then create a configuration file `ssl.conf` with this content:

```
ssl_protocols               TLSv1.2 TLSv1.3;
ssl_ecdh_curve              secp384r1;
ssl_ciphers                    "ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-
GCM-SHA512:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-
AES256-SHA384 OLD_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
OLD_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256";
ssl_prefer_server_ciphers   on;
ssl_dhparam                 /etc/nginx/dhparams.pem;
ssl_certificate             /etc/ssl/private/cert.pem;
ssl_certificate_key         /etc/ssl/private/key.pem;
ssl_session_timeout         10m;
ssl_session_cache           shared:SSL:10m;
ssl_session_tickets         off;
```

This contains all configuration variables related to the TLS connection. Note the `dhparams.pem` file that you just created. Make sure that you change the file names of the `ssl_certificate` and `ssl_certificate_key` files. The `/etc/ssl/private` directory in the container is `/home/pi/containers/certificates` on your Raspberry Pi.

Then finally create the main configuration file of nginx, `nginx.conf`:

```
worker_processes 1;

events { worker_connections 1024; }

http {

    sendfile on;
```

```
    server {
        listen 8091 ssl;
        include /etc/nginx/ssl.conf;

        location / {
            proxy_pass http://zwave2mqtt:8091;
            include    common_location.conf;
       }
    }

}
```

This is just an example if you want to access Zwave2Mqtt over HTTPS. In the `server` block, you let nginx listen on port 8091 with a TLS (`ssl`) connection, and you include the TLS configuration from `/etc/nginx/ssl.conf`. Then the block `location /` defines what happens when you visit the server's root directory. The `proxy_pass` directive says the request is passed on to port 8091 of the `zwave2mqtt` container over the HTTP protocol.

Now the only thing you need to add to make this work is the definition of the nginx container in your Docker Compose file. So go back to your home directory with cd and open `docker-compose.yml`. Add a definition for an nginx container:

```
version: '3.7'

services:
  # other services
  nginx:
    image: nginx:alpine
    container_name: nginx
    restart: always
    volumes:
      - ./containers/nginx:/etc/nginx
      - ./containers/certificates:/etc/ssl/private:ro
    ports:
      - "8091:8091"
  zwave2mqtt:
    image: robertslando/zwave2mqtt
    container_name: zwave2mqtt
    restart: always
    volumes:
      - ./containers/zwave2mqtt:/usr/src/app/store
    #ports:
    #  - "8091:8091"

    environment:
```

```
      - TZ=Europe/Brussels
    devices:
      - "/dev/ttyUSB0:/dev/ttyUSB0"

    sendfile on;
```

I have also added the definition of the zwave2mqtt container here to show what you should change there. I have commented out the original ports section because you have to remove it: you don't get direct access to Zwave2Mqtt's HTTP port anymore. Instead, the nginx container has this port 8091 exposed.

Warning:
If you don't remove or comment out the ports section of the zwave2mqtt container, your setup won't work because no two containers can listen on the same port of your Raspberry Pi.

After this, recreate your containers with:

```
docker-compose up -d
```

Now you can access the web interface of Zwave2Mqtt over TLS on https://HOST:8091 and not anymore on http://HOST:8091.

If you want to do the same for other services, just add another server block to your `nginx.conf`, with another port to listen on and another container name and port in the `proxy_pass` directive. Then add the port to expose to the ports section of your nginx container and remove or comment out the ports section of your other service. Recreate your containers with a `docker-compose up -d` and you're ready.

Note:
This is just a simple reverse proxy configuration. You can also configure nginx to listen on port 443 but take into account the hostname. If your local DNS server allows you to assign wildcard subdomains to your IP addresses (which all point to the same IP address), you can then visit https://zwave.pi.home to be forwarded to Zwave2Mqtt, https://node-red.pi.home to be forwarded to Node-RED and so on. This is beyond the scope of this book.

15.5.2 ● Adding basic authentication to nginx

Now that Zwave2Mqtt isn't exposed directly anymore to the network but has to be accessed via nginx, you can add other protection measures, such as a username and password. To be able to create a password hash, you have to install the package `apache2-utils` first on

Raspberry Pi OS:

```
sudo apt install apache2-utils
```

Afterwards, create a password file and add a user with:

```
htpasswd -c /home/pi/containers/nginx/.htpasswd USER
```

Add your preferred username instead of `USER`.

The program then asks you to enter your password. Make sure to choose a strong password (see Chapter 3). Retype it. After this, your user is added to the file, together with a hash of the password. If you want to create a second user, just enter the command again, but without the `-c` option (which creates a new password file) and with another username.

The output of the file should look something like this:

```
koan:$apr1$oBgoppc6$kO14XotRBC8hTnXcu3H8N0
testuser:$apr1$njYvKGLp$3GSy87JMQBh5fRWRv4TRt/
```

Each line has a user, a colon, a hash type between dollar signs, and then the hashed password.

Now open your `nginx.conf` and add the following lines to the `location` block of the service you want to secure, in this example Zwave2Mqtt:

```
auth_basic "Zwave2Mqtt";
auth_basic_user_file /etc/nginx/.htpasswd;
```

Then restart the nginx container:

```
docker restart nginx
```

If you now visit the URL in your web browser, it asks you to enter a valid username and password. If the combination is wrong, you'll see a HTTP 401 Authorization Required error page.

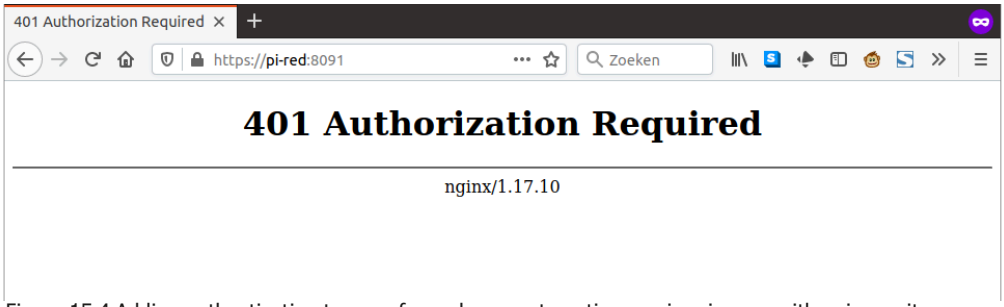

Figure 15.4 Adding authentication to one of your home automation services is easy with nginx as its reverse proxy.

15.6 • Bridging two MQTT brokers securely

If anyone in your network can just listen to MQTT messages or sniff unencrypted MQTT traffic, this can leave some private data exposed. Especially with Rhasspy, because it streams your voice over the network and publishes sentences it recognizes to MQTT.
In this book, I advocate the use of encryption and authentication as much as possible, also for MQTT. However, some projects in this book don't support (yet) a TLS connection to an MQTT broker. This doesn't mean that you have to use them in an insecure way. There's a solution.

For all containers running on the same Raspberry Pi as your MQTT broker, you can use an unencrypted connection to mosquitto by using the container's name. The network traffic then stays on your Raspberry Pi, because it uses Docker's internal network. So there's no real need to encrypt the traffic.

The problem only starts when you're running one of your services on another machine and they don't support MQTT over TLS. However, there's an easy fix: run another mosquitto container on that second machine and let the service talk to this container, unencrypted over that machine's internal Docker network. Then bridge this mosquitto container to your main MQTT broker, over an encrypted connection.

I'll illustrate this with rtl_433 (see Chapter 7), but you can use the same method for other services.

This setup doesn't need any change on your main MQTT broker. On the second machine, add a mosquitto container to your `docker-compose.yml` file:

```
version: '3.7'

services:
  # other services
  mosquitto:
    image: eclipse-mosquitto
    container_name: mosquitto
    restart: always
    volumes:
      - ./containers/mosquitto/config:/mosquitto/config
      - ./containers/mosquitto/data:/mosquitto/data
      - ./containers/mosquitto/log:/mosquitto/log
      - ./containers/certificates:/mosquitto/config/certs:ro
      - /etc/localtime:/etc/localtime:ro
    user: "1000:1000"
```

Note that you don't expose any ports in this service definition: only the other containers on this machine can connect (using Docker's internal network) to this broker.

Now create some directories for Mosquitto's configuration, data and logs:

```
mkdir -p /home/pi/containers/mosquitto/{config,data,log}
```

Create a configuration file for mosquitto:

```
nano /home/pi/containers/mosquitto/config/mosquitto.conf
```

Then put the basic configuration for mosquitto (see Chapter 4) in this file, and add some extra configuration directives to create a bridge:

```
port 1883
listener 9001
protocol websockets
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log

connection bridge
address HOSTNAME:8883
remote_username home
remote_password PASSWORD
```

```
bridge_cafile /mosquitto/config/certs/rootCA.pem
bridge_insecure false
topic # both
```

The first part is just the basic configuration for mosquitto, so it listens to port 1883 and (for the WebSocket protocol) port 9001, unencrypted. You can always add authentication if you want, see Chapter 4 for the details.

Then the part beginning with `connection bridge` configures a bridge, with the hostname and port number of your main MQTT broker, as well as a valid username and password. Make sure that your mosquitto container can resolve this address, so make it a fully qualified domain name such as `pi-red.home`. You also specify the root CA file to verify the broker's TLS certificate.

The last line is as simple as it's magical. This says that every topic (the # wildcard) that is coming in or out of is forwarded between the main MQTT broker and this MQTT broker.

Save this file, and then create a directory for the root CA file:

```
mkdir -p /home/pi/containers/certificates
```

And then copy the root CA file from your main home automation gateway to this directory. For instance:

```
scp pi-red:containers/certificates/rootCA.pem containers/certificates
```

After this, you can recreate your containers:

```
docker-compose up -d
```

And now you can connect the containers on this machine to your local MQTT broker, which communicates transparently over TLS to your main MQTT broker. No unencrypted data leave your machine. Rtl_433 connects to the local MQTT broker unencrypted and thanks to the encrypted bridge is also connected transparently to the main MQTT broker.

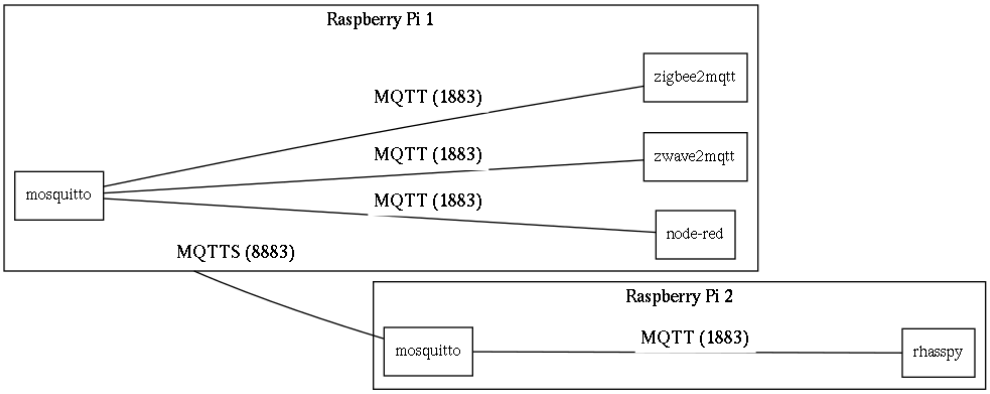

Figure 15.5 With two Raspberry Pis running a mosquitto container in an encrypted MQTT bridge configuration, the first one's internal MQTT traffic doesn't leave its Raspberry Pi unencrypted.

• Index

elektorbooks

Control Your Home with Raspberry Pi

Secure, Modular, Open-Source and Self-Sufficient

Ever since the Raspberry Pi was introduced, it has been used by enthusiasts to automate their homes. The Raspberry Pi is a powerful computer in a small package, with lots of interfacing options to control various devices. This book shows you how you can automate your home with a Raspberry Pi. You'll learn how to use various wireless protocols for home automation, such as Bluetooth, 433.92 MHz radio waves, Z-Wave, and Zigbee. Soon you'll automate your home with Python, Node-RED, and Home Assistant, and you'll even be able to speak to your home automation system. All this is done securely, with a modular system, completely open-source, without relying on third-party services. You're in control of your home, and no one else.

At the end of this book, you can install and configure your Raspberry Pi as a highly flexible home automation gateway for protocols of your choice, and link various services with MQTT to make it your own system. This DIY (do it yourself) approach is a bit more laborious than just installing an off-the-shelf home automation system, but in the process, you can learn a lot, and in the end, you know exactly what's running your house and how to tweak it. This is why you were interested in the Raspberry Pi in the first place, right?

- Turn your Raspberry Pi into a reliable gateway for various home automation protocols.
- Make your home automation setup reproducible with Docker Compose.
- Secure all your network communication with TLS.
- Create a video surveillance system for your home.
- Automate your home with Python, Node-RED, Home Assistant and AppDaemon.
- Securely access your home automation dashboard from remote locations.
- Use fully offline voice commands in your own language.

Koen Vervloesem has been writing for over 20 years on Linux, open-source software, security, home automation, AI, and programming. He holds a Master's degree in Computer Science Engineering, a Master's degree in Philosophy and an LPIC-3 303 Security certificate. He is editor-in-chief of the Dutch MagPi magazine and is a board member of the Belgian privacy activist organization, the Ministry of Privacy.

Elektor International Media BV
www.elektor.com

ISBN 978-1-907920-94-3
9 781907 920943

elektor
design > share > sell